On BigPipe
On Node.js

by @undoZen

What I'll talk today

What I'll talk today

- on BigPipe - What's it?

What I'll talk today

- on BigPipe - What's it?
- on "BigPipe on Node" - How to implement it?

What I'll talk today

- on BigPipe - What's it?
- on "BigPipe on Node" - How to implement it?
- on Node - Share some personal thoughts on node.

On BigPipe

What’s BigPipe

What’s BigPipe

- It’s a technique invented by Facebook to improve web page loading performance

What’s BigPipe

- It’s a technique invented by Facebook to improve web page loading performance
- Server

What’s BigPipe

- It’s a technique invented by Facebook to improve web page loading performance
- Server
  - Transfer document using chunked encoding

Chunked Encoding

Chunked Encoding

- Introduced by HTTP/1.1 in 1999

Chunked Encoding

- Introduced by HTTP/1.1 in 1999
- In HTTP/1.0 Content-Length header is required
  - Size of HTTP body transfered before it

Chunked Encoding

- Introduced by HTTP/1.1 in 1999
- In HTTP/1.0 Content-Length header is required
  - Size of HTTP body transfered before it
- In HTTP/1.1 you have Transfer-Encoding: chunked
  - a chunk include a piece of response data and it's length before it
  - an empty chunk indicate the end of transferring

Content-Length vs Chunked Encoding

```
var http = require('http');
http.createServer(function (req, res) {
  var msg = '<!doctype html><h1>Hello</h1><h2>World</h2>';
  res.setHeader('Content-Type', 'text/html');
  res.setHeader('Content-Length', Buffer.byteLength(msg));
  res.end(msg);
}).listen(1234);
```

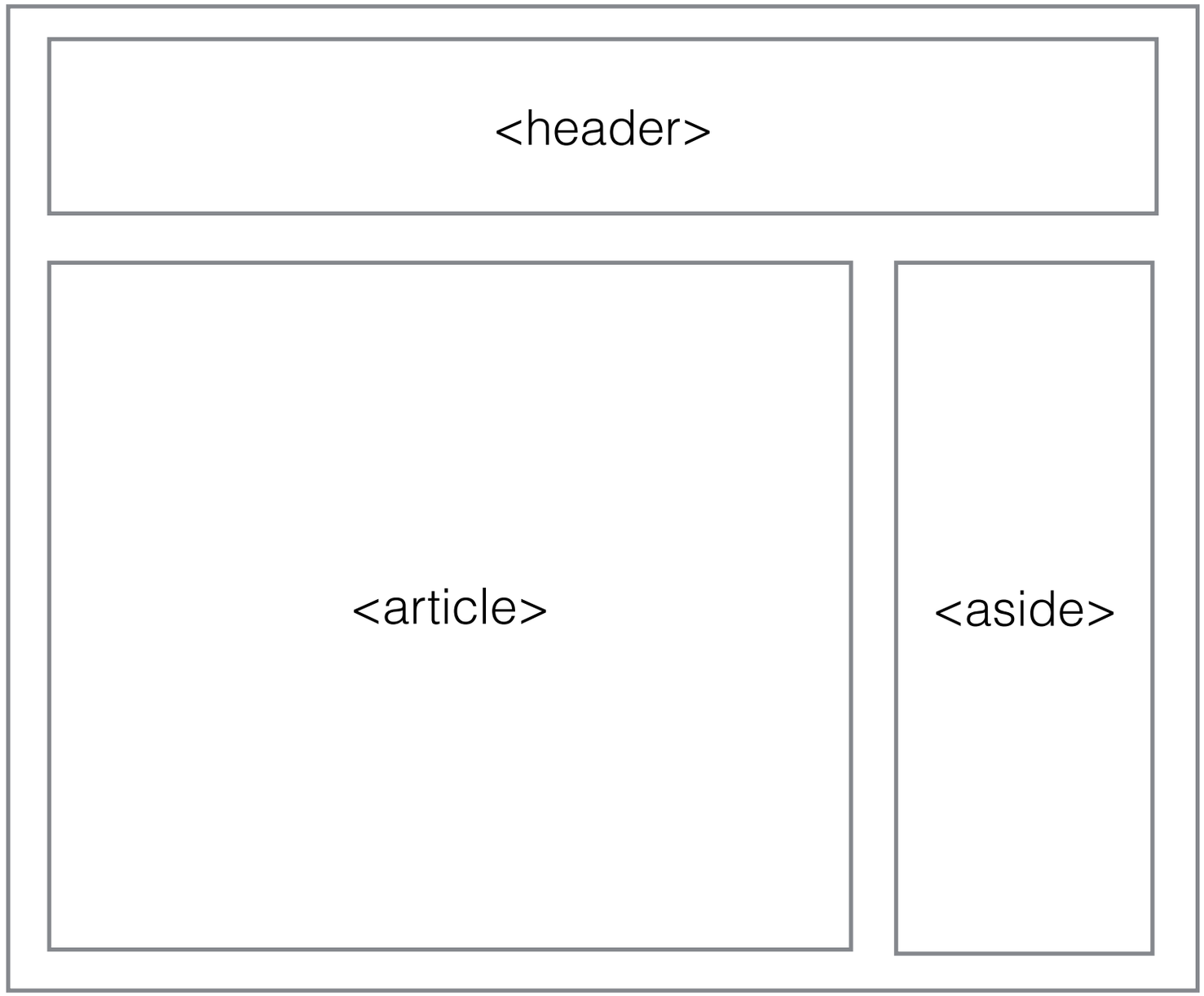
```
var http = require('http');
http.createServer(function (req, res) {
  res.setHeader('Content-Type', 'text/html');
  res.write('<!doctype html>');
  res.write('<h1>Hello</h1>');
  res.write('<h2>World</h2>');
  res.end();
}).listen(1234);
```

```
3. telnet (40,16)
~ - @undozen
$ telnet 127.0.0.1 1234
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
GET / HTTP/1.1
Host: localhost

HTTP/1.1 200 OK
Content-Type: text/html
Content-Length: 43
Date: Fri, 08 Nov 2013 15:36:08 GMT
Connection: keep-alive

<!doctype html><h1>Hello</h1><h2>World</
h2>
```

```
3.
~ - @undozen
$ telnet 127.0.0.1 1234
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
GET / HTTP/1.1
Host: localhost

HTTP/1.1 200 OK
Content-Type: text/html
Date: Fri, 08 Nov 2013 02:02:04 GMT
Connection: keep-alive
Transfer-Encoding: chunked

f
<!doctype html>
e
<h1>Hello</h1>
e
<h2>World</h2>
0
```

Go on …

Here’s a simple web page…



… and it’s simple HTML

```
<!doctype html>
<html>
…
<body>

  <header>
    {{header_content}}
  </header>

  <article>
   {{article_content}}
  </article>

  <aside>
    {{aside_content}}
  </aside>

<footer>…</footer>
</body>
</html>
```

… and it’s simple HTML

```
<!doctype html>
<html>
…
<body>

  <header>
    {{header_content}}
  </header>

  <article>
   {{article_content}}
  </article>

  <aside>
    {{aside_content}}
  </aside>

<footer>…</footer>
</body>
</html>
```

- Question:
  - How we transfer it using chunked encoding?

… and it’s simple HTML

```
<!doctype html>
<html>
…
<body>

  <header>
    {{header_content}}
  </header>

  <article>
   {{article_content}}
  </article>

  <aside>
    {{aside_content}}
  </aside>

<footer>…</footer>
</body>
</html>
```

- Question:
  - How we transfer it using chunked encoding?
- Answer:
  - Send it chunk by chunk.

… and it’s simple HTML

- Question:
  - How we transfer it using chunked encoding?
- Answer:
  - Send it chunk by chunk.

… and it’s simple HTML

Chunk #1

```
<!doctype html>
<html>
…
<body>
```

#2

```
<header>
  {{header_content}}
</header>
```

#3

```
<article>
 {{article_content}}
</article>
```

#4

```
<aside>
  {{aside_content}}
</aside>
```

#5

```
<footer>…</footer>
</body>
</html>
```

- Question:
  - How we transfer it using chunked encoding?
- Answer:
  - Send it chunk by chunk.

… and it’s simple HTML

Chunk #1

```
<!doctype html>
<html>
…
<body>
```

#2

```
<header>
  {{header_content}}
</header>
```

#3

```
<article>
 {{article_content}}
</article>
```

#4

```
<aside>
  {{aside_content}}
</aside>
```

#5

```
<footer>…</footer>
</body>
</html>
```

- Question:
  - How we transfer it using chunked encoding?
- Answer:
  - Send it chunk by chunk.

… and it’s simple HTML

Chunk #1

```
<!doctype html>
<html>
…
<body>
```

#2

```
<header>
  {{header_content}}
</header>
```

#3

```
<article>
 {{article_content}}
</article>
```

#4

```
<aside>
  {{aside_content}}
</aside>
```

#5

```
<footer>…</footer>
</body>
</html>
```

- Question:
  - How we transfer it using chunked encoding?
- Answer:
  - Send it chunk by chunk.

… and it’s simple HTML

Chunk #1

```
<!doctype html>
<html>
…
<body>
```

#2

```
<header>
  {{header_content}}
</header>
```

#3

```
<article>
 {{article_content}}
</article>
```

#4

```
<aside>
  {{aside_content}}
</aside>
```

#5

```
<footer>…</footer>
</body>
</html>
```

- Question:
  - How we transfer it using chunked encoding?
- Answer:
  - Send it chunk by chunk.

… and it’s simple HTML

Chunk #1

```
<!doctype html>
<html>
…
<body>
```

#2

```
<header>
  {{header_content}}
</header>
```

#3

```
<article>
 {{article_content}}
</article>
```

#4

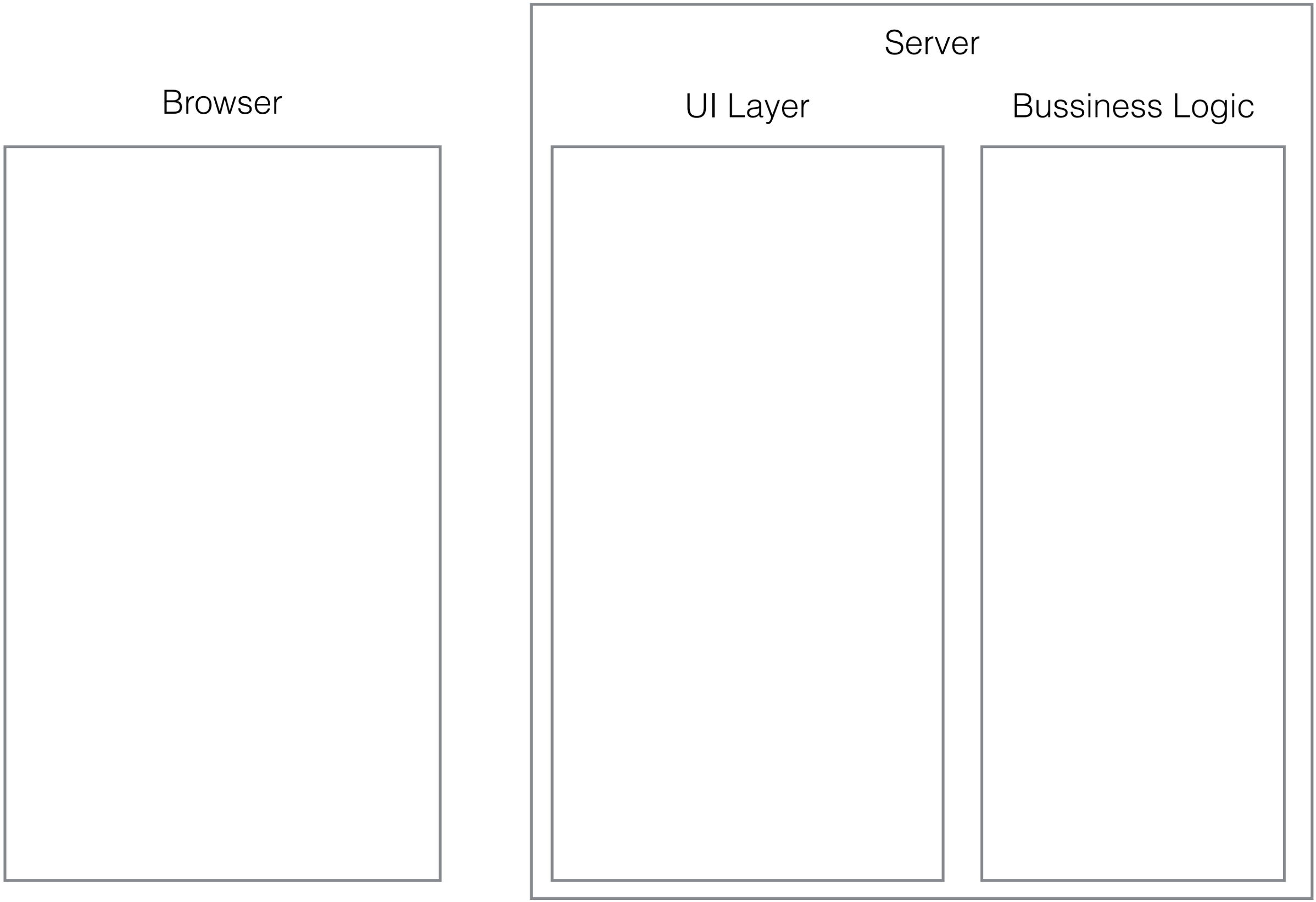
```
<aside>
  {{aside_content}}
</aside>
```

#5

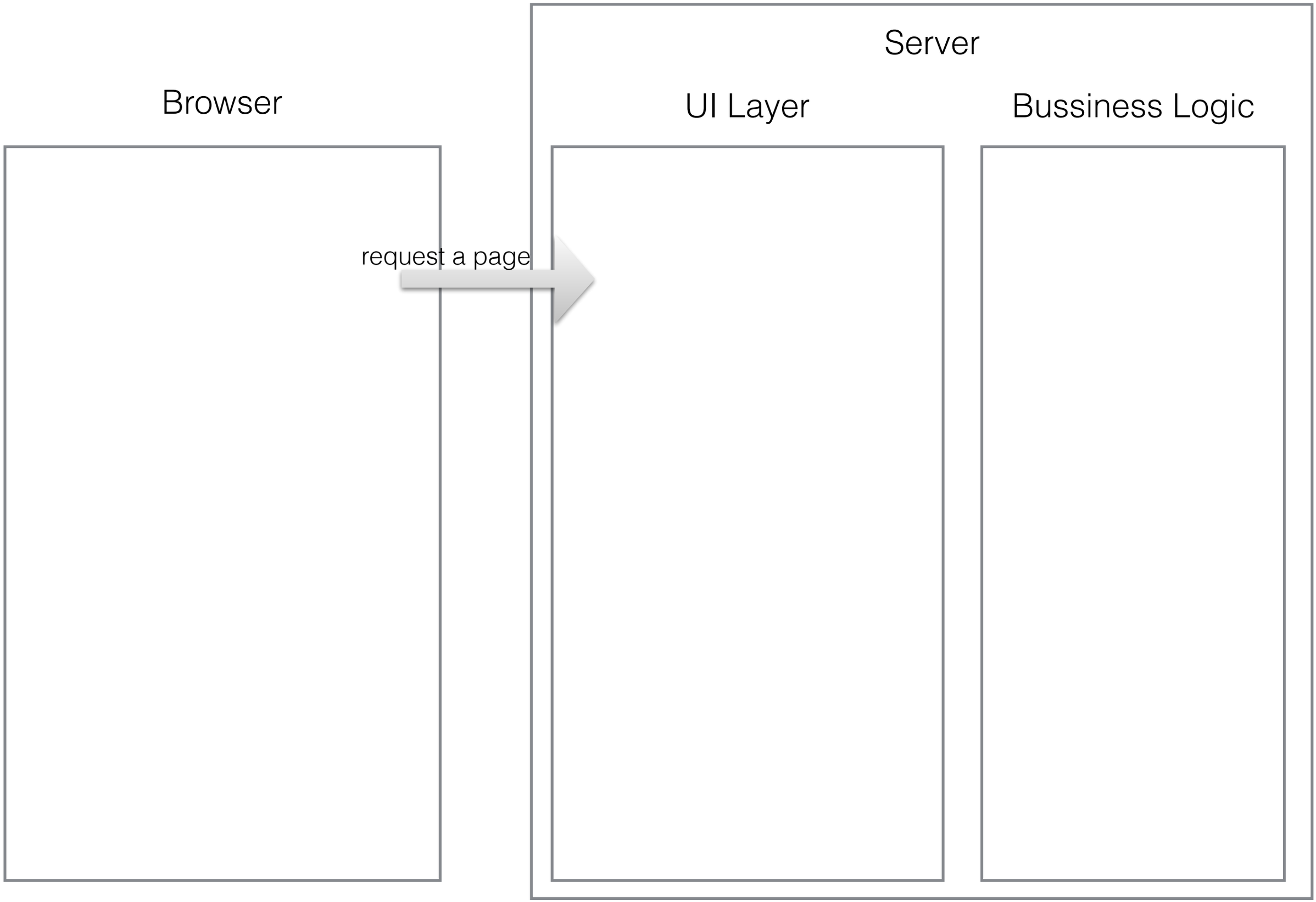
```
<footer>…</footer>
</body>
</html>
```

- Question:
  - How we transfer it using chunked encoding?
- Answer:
  - Send it chunk by chunk.

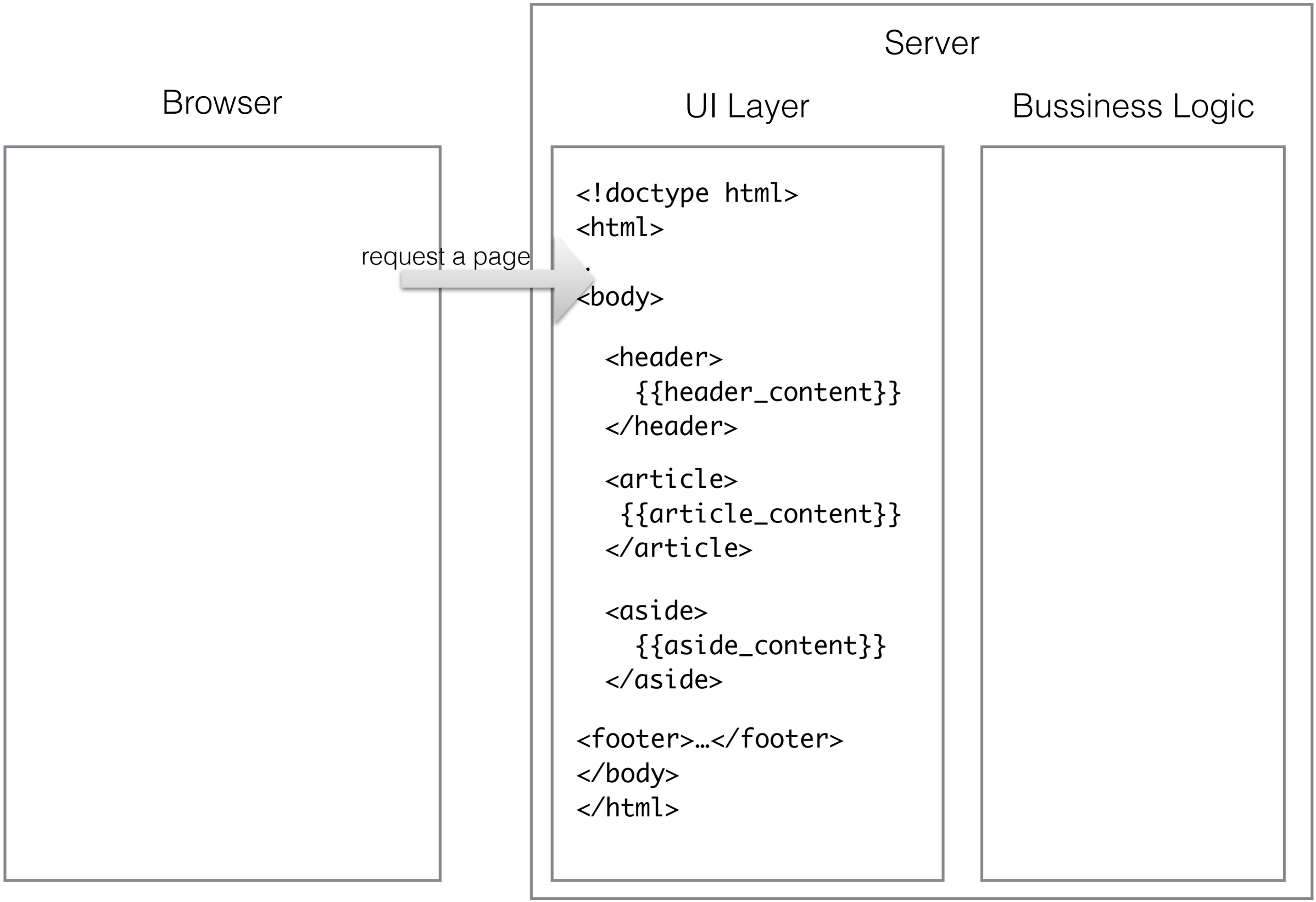
The Traditional Way of Web Page Generating



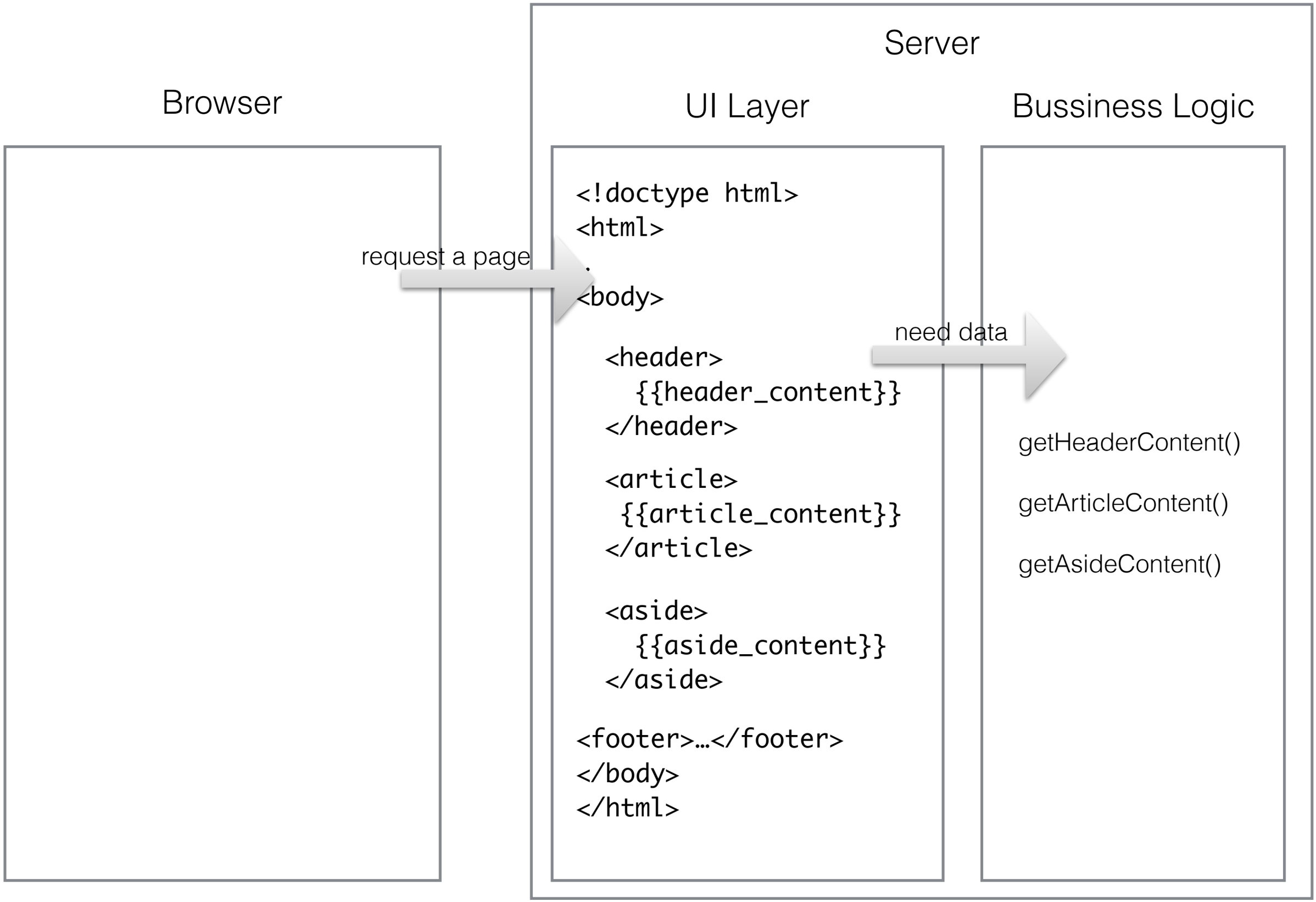
The Traditional Way of Web Page Generating



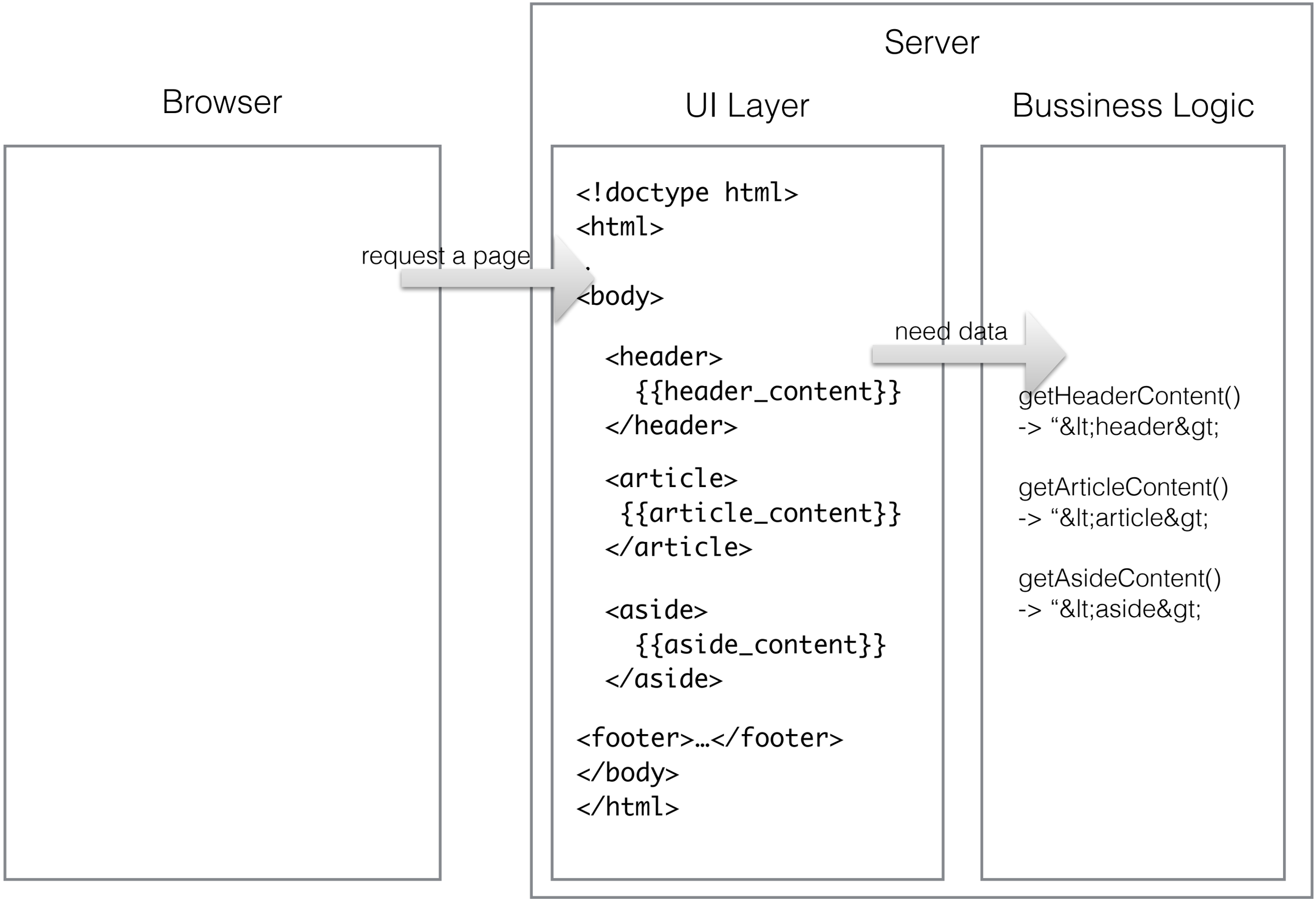
The Traditional Way of Web Page Generating



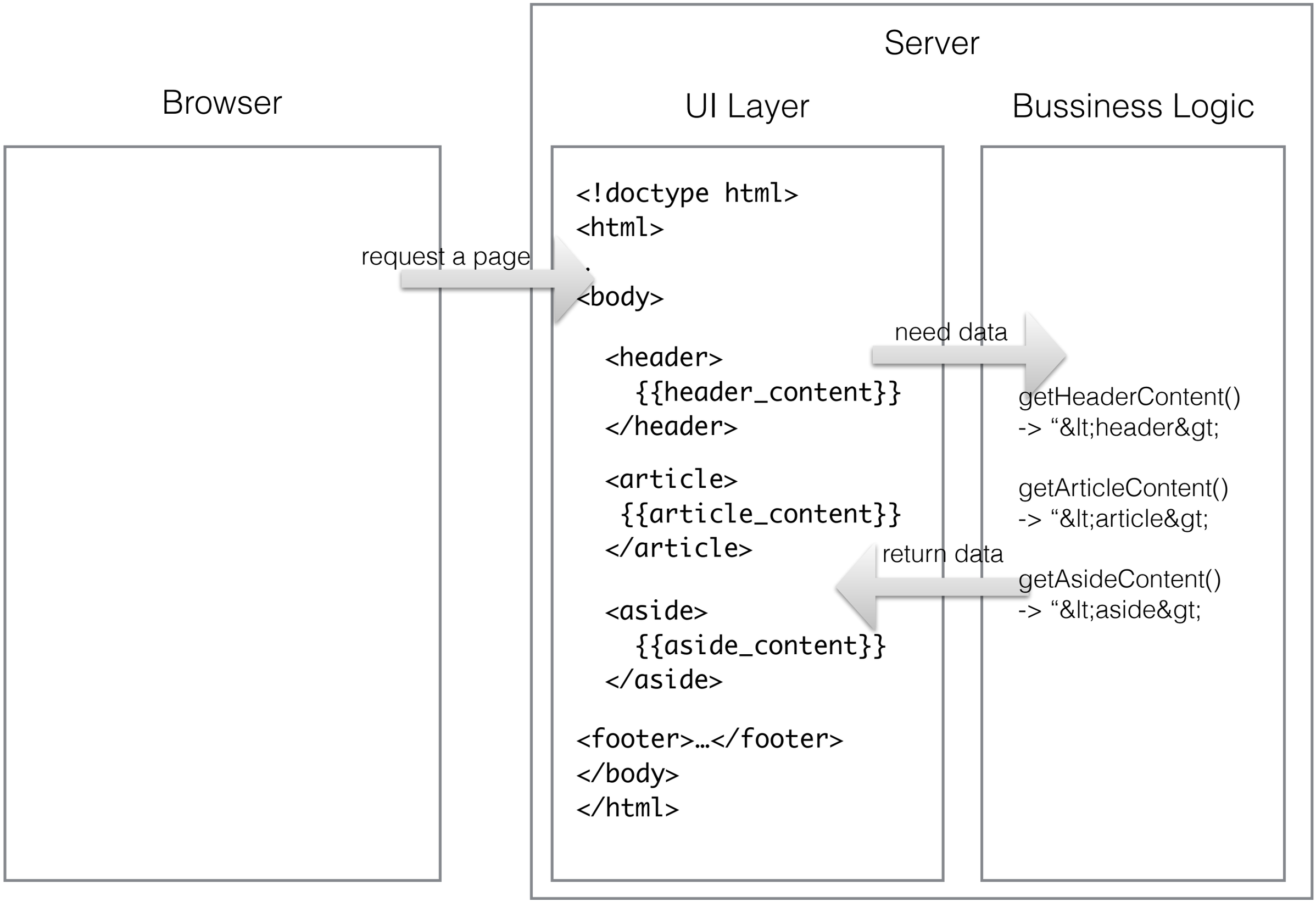
The Traditional Way of Web Page Generating



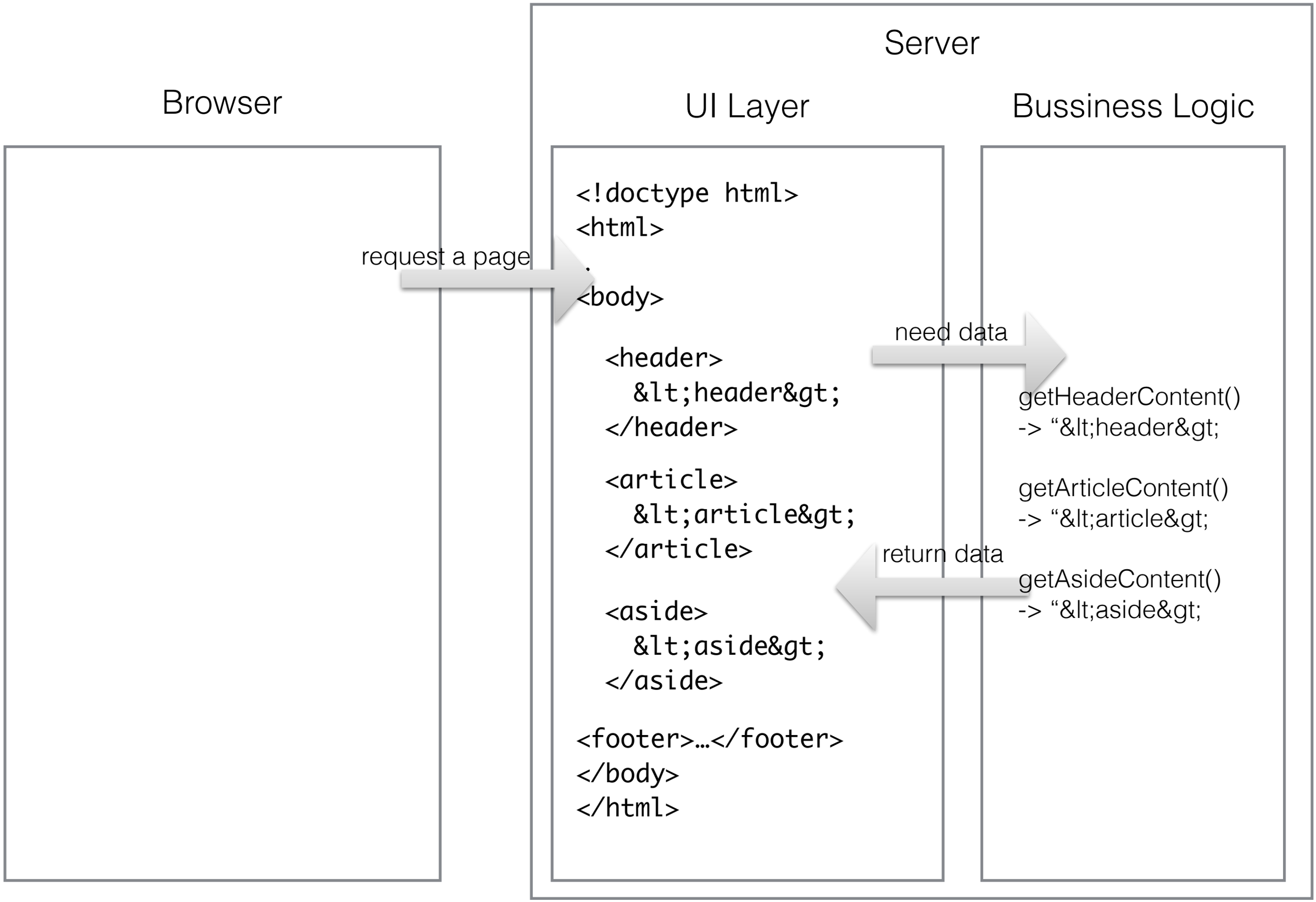
The Traditional Way of Web Page Generating



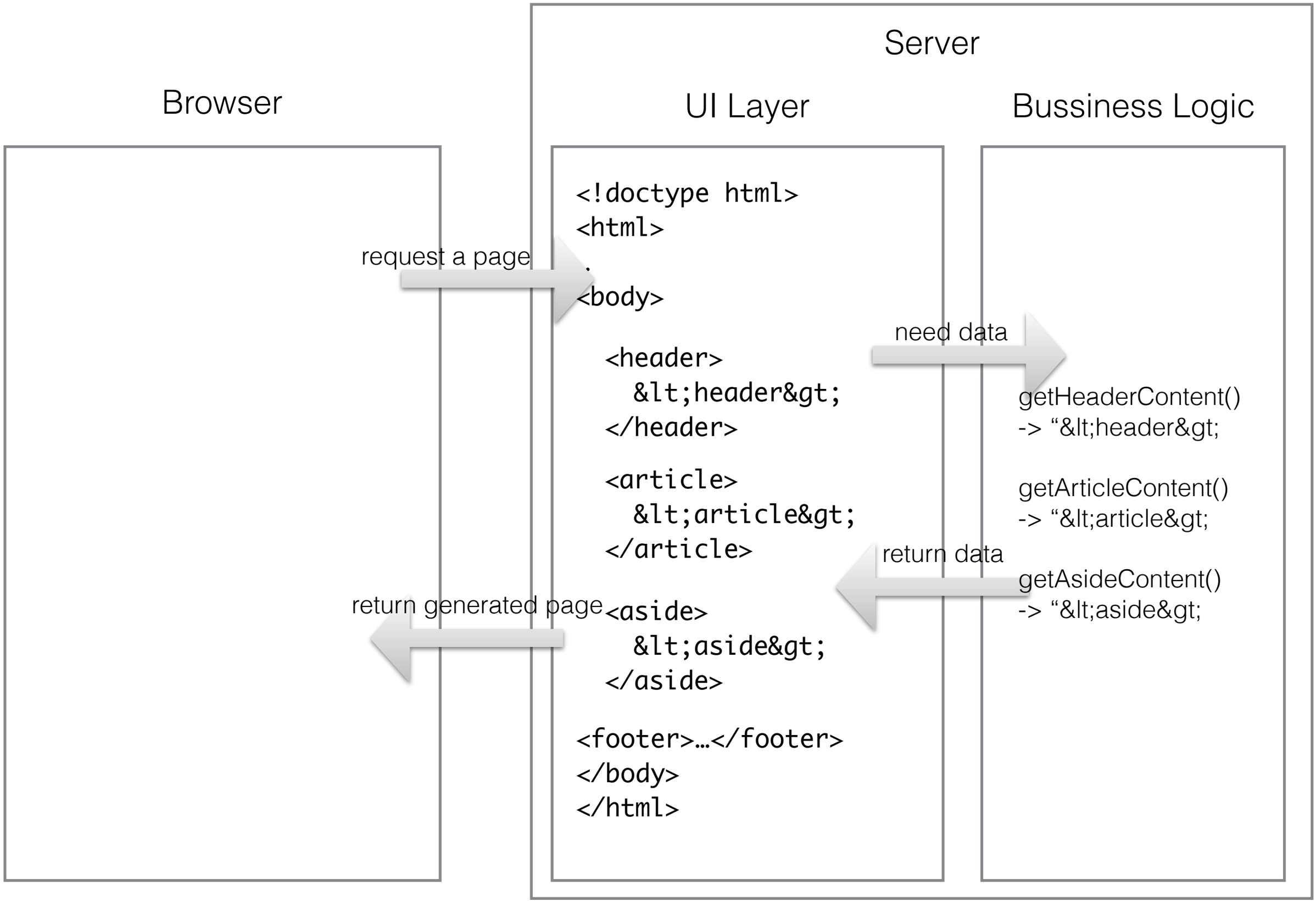
The Traditional Way of Web Page Generating



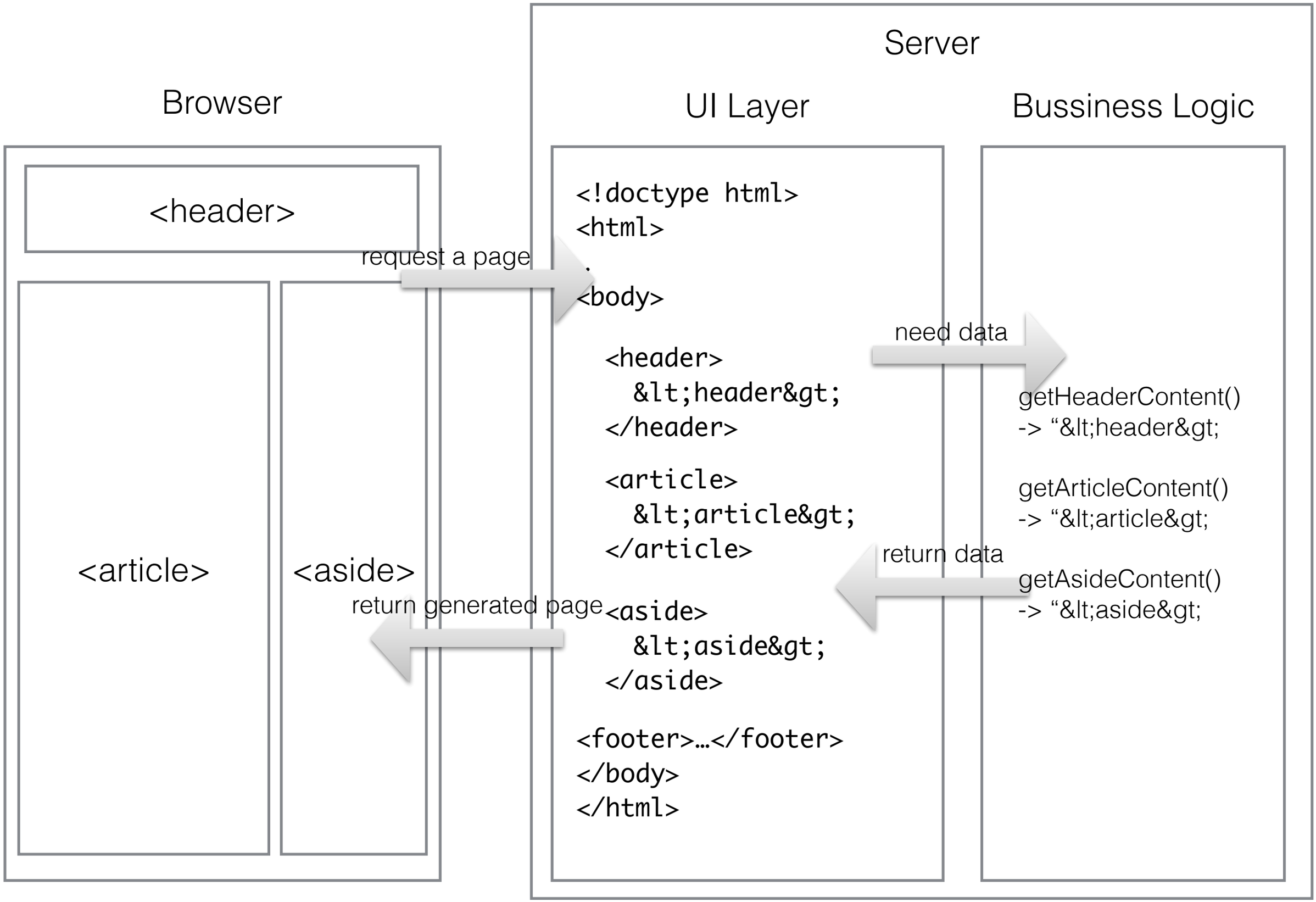
The Traditional Way of Web Page Generating



The Traditional Way of Web Page Generating



The Traditional Way of Web Page Generating



Simplify the generating process

Simplify the generating process

- Bussiness Logic has data, UI Layer has template

Simplify the generating process

- Bussiness Logic has data, UI Layer has template
- Server assemble the data and template, generate entire HTML document to send to the browser

Simplify the generating process

- Bussiness Logic has data, UI Layer has template
- Server assemble the data and template, generate entire HTML document to send to the browser
- Browser receive the document, then parse and render it

Zoom In Bussiness Logic Part…

Zoom In Bussiness Logic Part…

- Assume that in the bussiness logic
  - getHeaderContent() taks 500ms
  - getArticleContent() takes 100ms
  - getAsideContent() takes 300ms

Zoom In Bussiness Logic Part…

- Assume that in the bussiness logic
  - getHeaderContent() taks 500ms
  - getArticleContent() takes 100ms
  - getAsideContent() takes 300ms
- We need to wait for getHeaderContent() to finish

Zoom In Bussiness Logic Part…

- Assume that in the bussiness logic
  - getHeaderContent() taks 500ms
  - getArticleContent() takes 100ms
  - getAsideContent() takes 300ms
- We need to wait for getHeaderContent() to finish
- Because we don’t want HTML source messed

Like this…

Like this…

```
<!doctype html>
<html>
…
<body>
```

Like this…

```
<!doctype html>
<html>
…
<body>
  <article>
    &lt;article&gt;
  </article>
```

100ms data

Like this…

```
<!doctype html>
<html>
…
<body>
  <article>  100ms data
    &lt;article&gt;
  </article>
  <aside>  300ms data
    &lt;aside&gt;
  </aside>
```

Like this…

```
<!doctype html>
<html>
…
<body>
  <article>  100ms data
    &lt;article&gt;
  </article>

  <aside>  300ms data
    &lt;aside&gt;
  </aside>

  <header>  500ms data
    &lt;header&gt;
  </header>
```

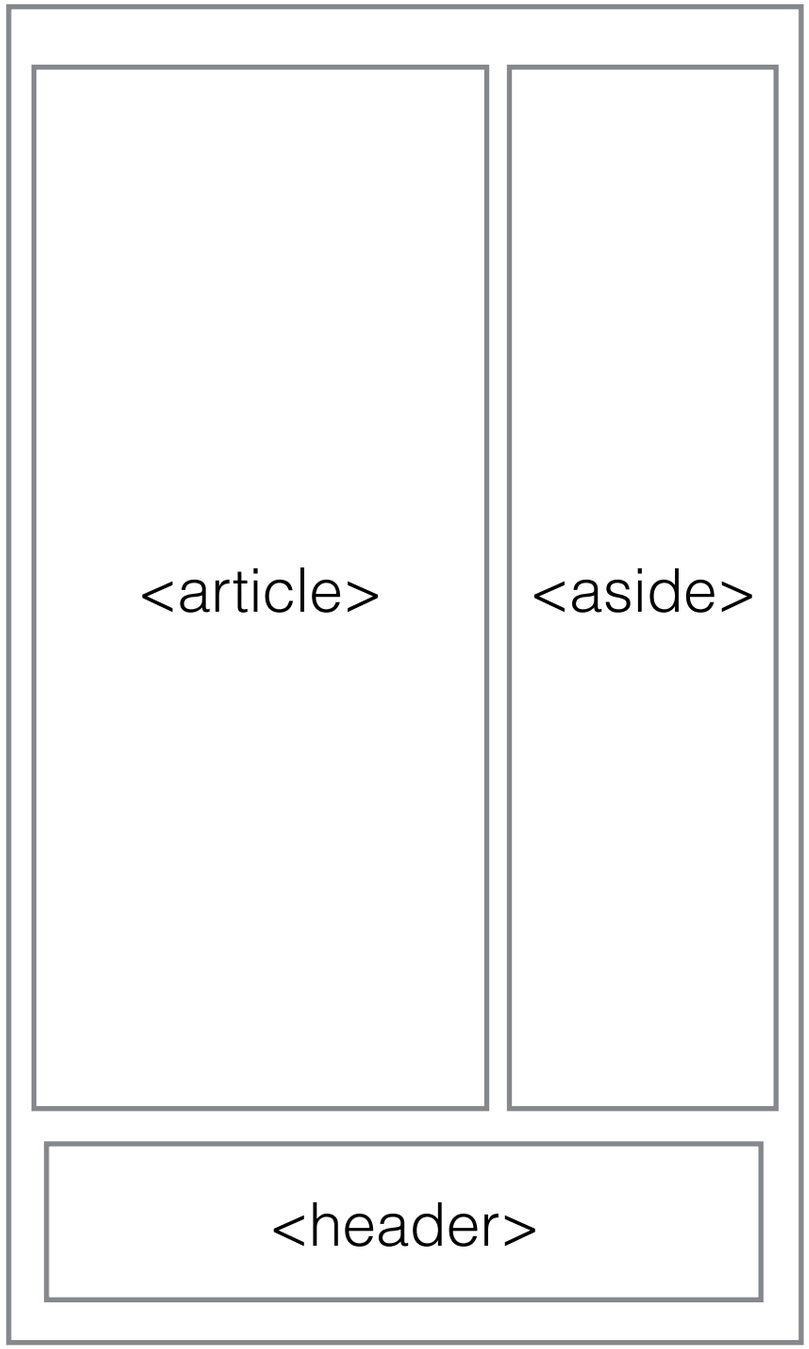
Like this…

```
<!doctype html>
<html>
…
<body>

  <article>  100ms data
    &lt;article&gt;
  </article>

  <aside>  300ms data
    &lt;aside&gt;
  </aside>

  <header>  500ms data
    &lt;header&gt;
  </header>

<footer>…</footer>
</body>
</html>
```

Like this…

```
<!doctype html>
<html>
…
<body>

  <article>  100ms data
    &lt;article&gt;
  </article>

  <aside>  300ms data
    &lt;aside&gt;
  </aside>

  <header>  500ms data
    &lt;header&gt;
  </header>

<footer>…</footer>
</body>
</html>
```

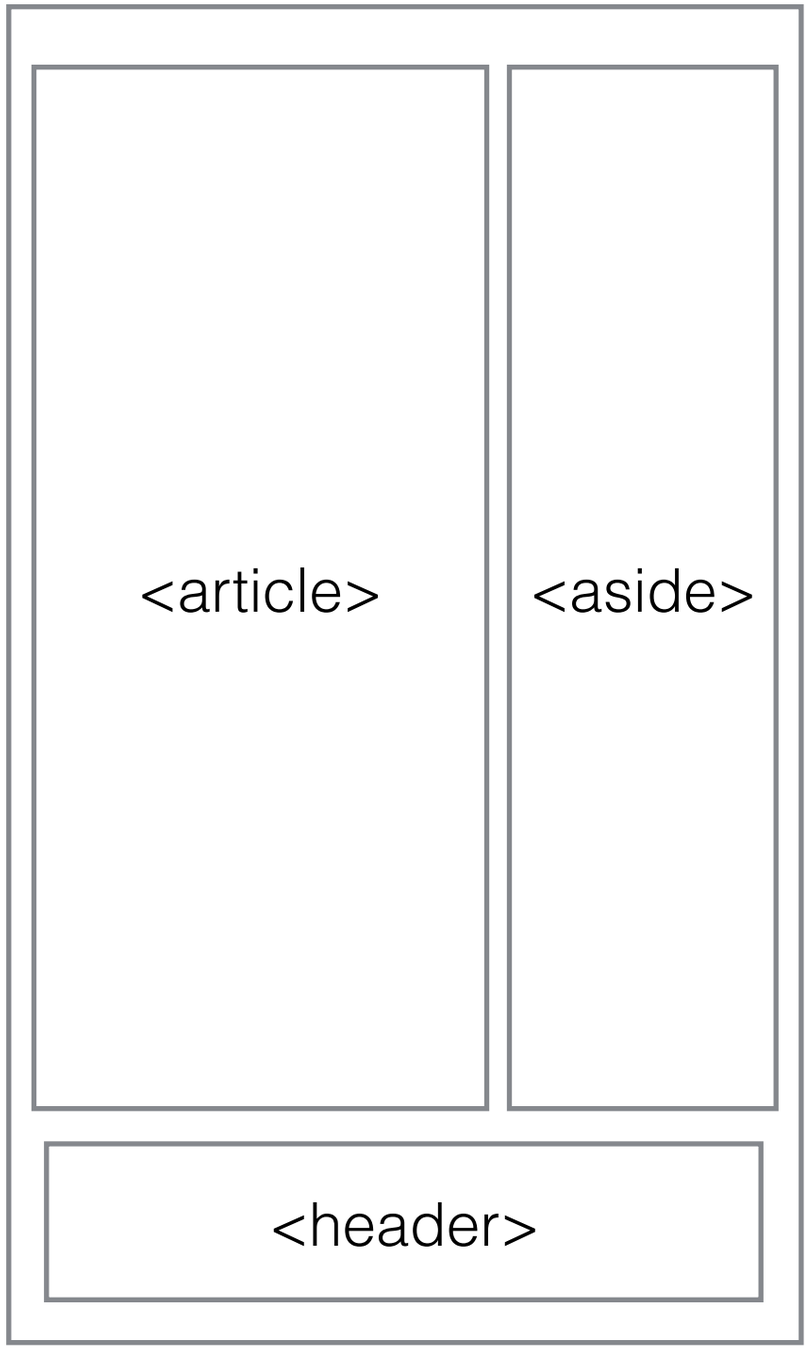
So we will get this in browser



Like this…

```
<!doctype html>
<html>
…
<body>

  <article>  100ms data
    &lt;article&gt;
  </article>

  <aside>  300ms data
    &lt;aside&gt;
  </aside>

  <header>  500ms data
    &lt;header&gt;
  </header>

<footer>…</footer>
</body>
</html>
```

So we will get this in browser

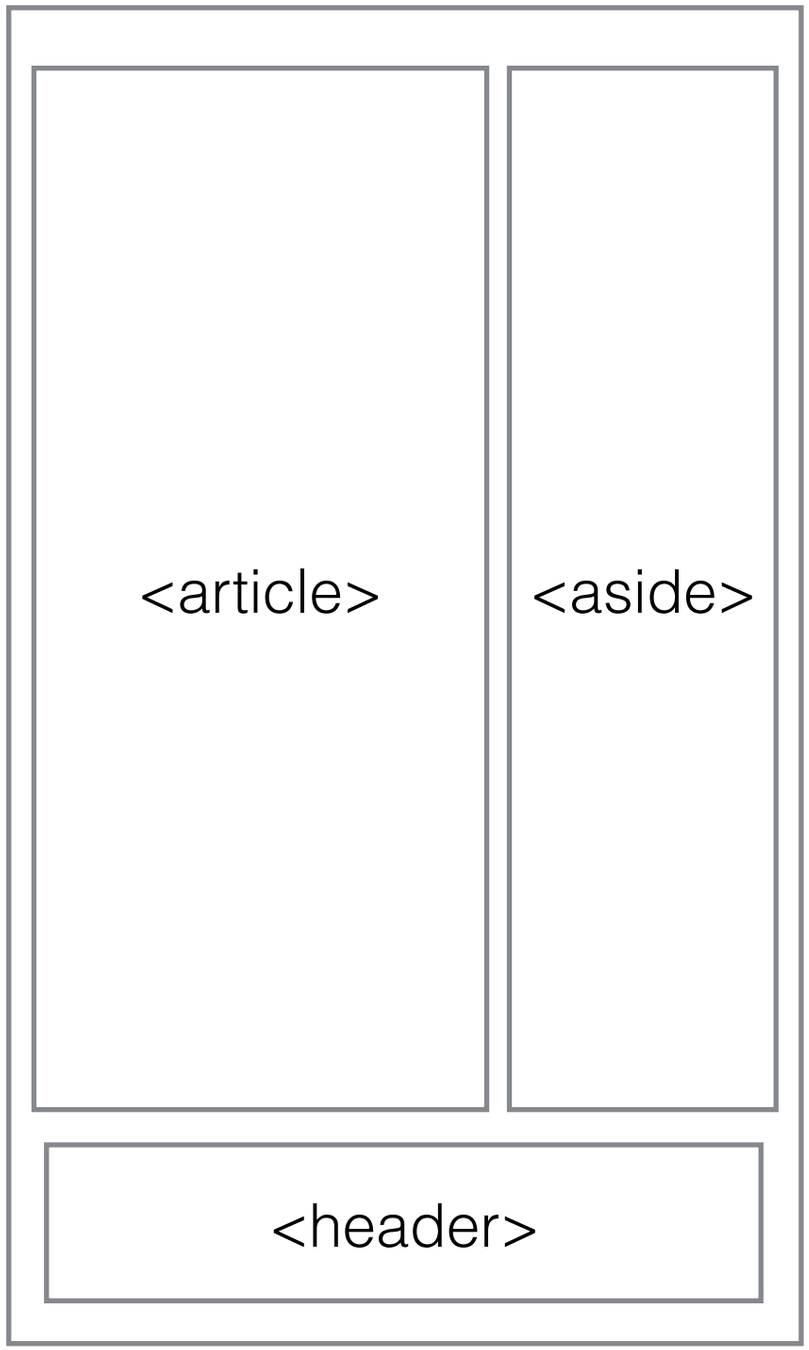


- So in this way chunked encoding doesn't meaning much to us.

Like this…

```
<!doctype html>
<html>
…
<body>

  <article>  100ms data
    &lt;article&gt;
  </article>

  <aside>  300ms data
    &lt;aside&gt;
  </aside>

  <header>  500ms data
    &lt;header&gt;
  </header>

<footer>…</footer>
</body>
</html>
```

So we will get this in browser



- So in this way chunked encoding doesn't meaning much to us.
- Because the order matters.

What’s BigPipe

- It’s a technique invented by Facebook to improve web page loading performance.
- Server
  - Transfer document using chunked encoding
  - Transferring order of chunked data doesn’t matter

How?

We have to question that…

We have to question that…

- Does HTML really need to be assembled in server-side?

We have to question that…

- Does HTML really need to be assembled in server-side?
- Of Course Not!

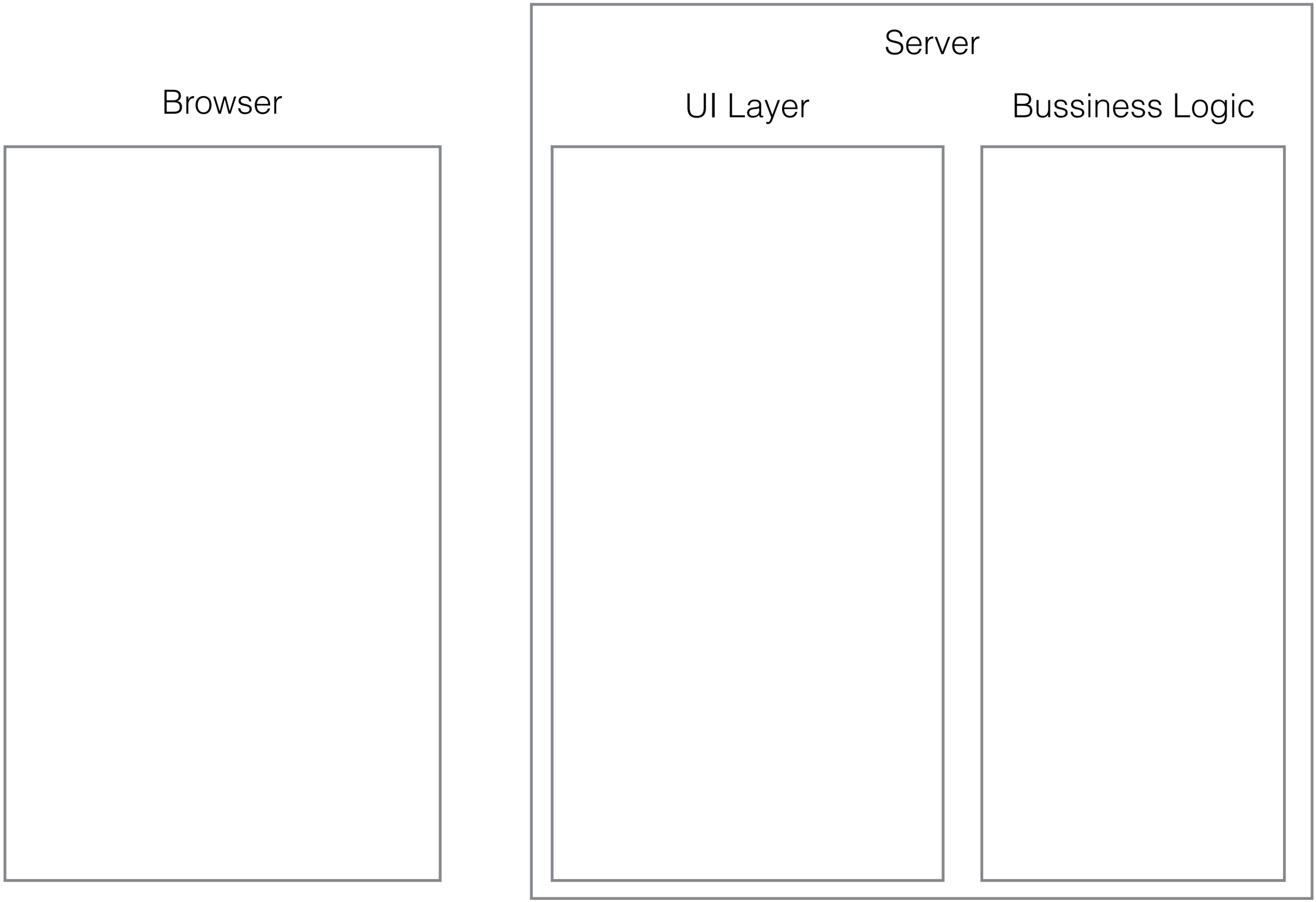
We have to question that…

- Does HTML really need to be assembled in server-side?
- Of Course Not!
- There’s client-side templating everywhere now

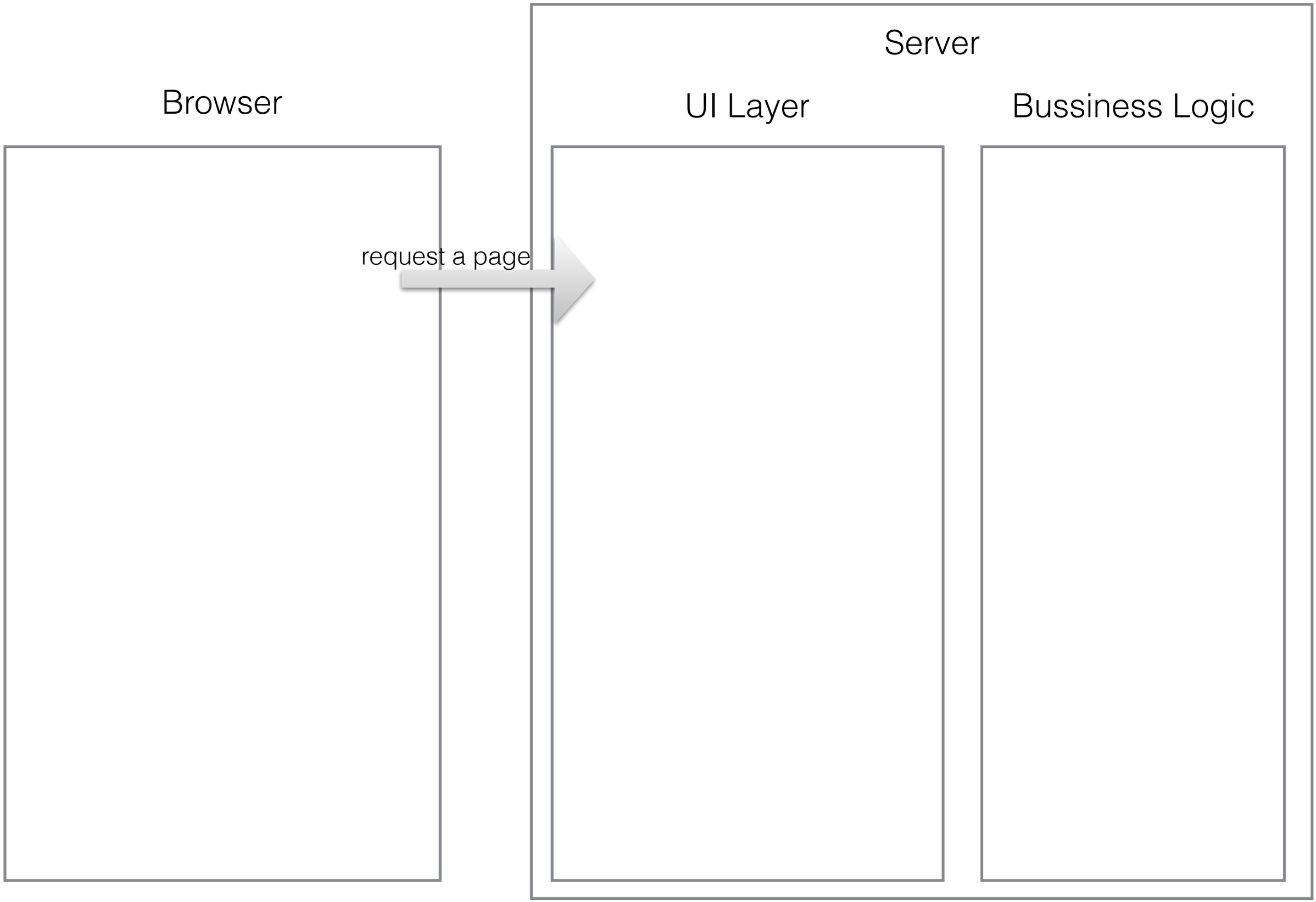
What’s BigPipe

- It’s a technique invented by Facebook to improve page loading performance.
- Server
  - Transfer document using chunked encoding
  - Chunked data transferring order doesn’t matter
- Browser
  - Initialize web page using JavaScript

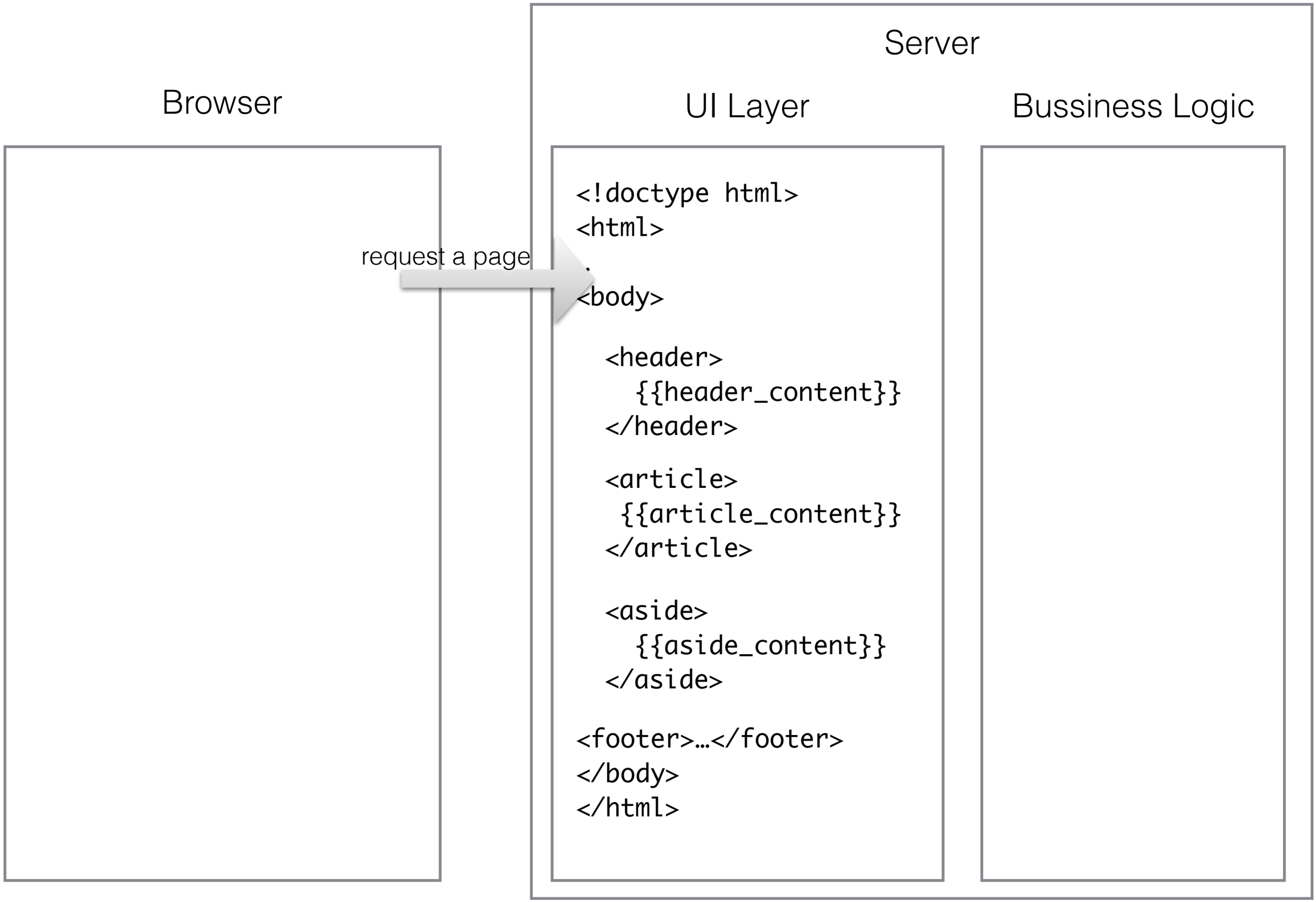
So Here's the BigPipe Way of Page Initialization



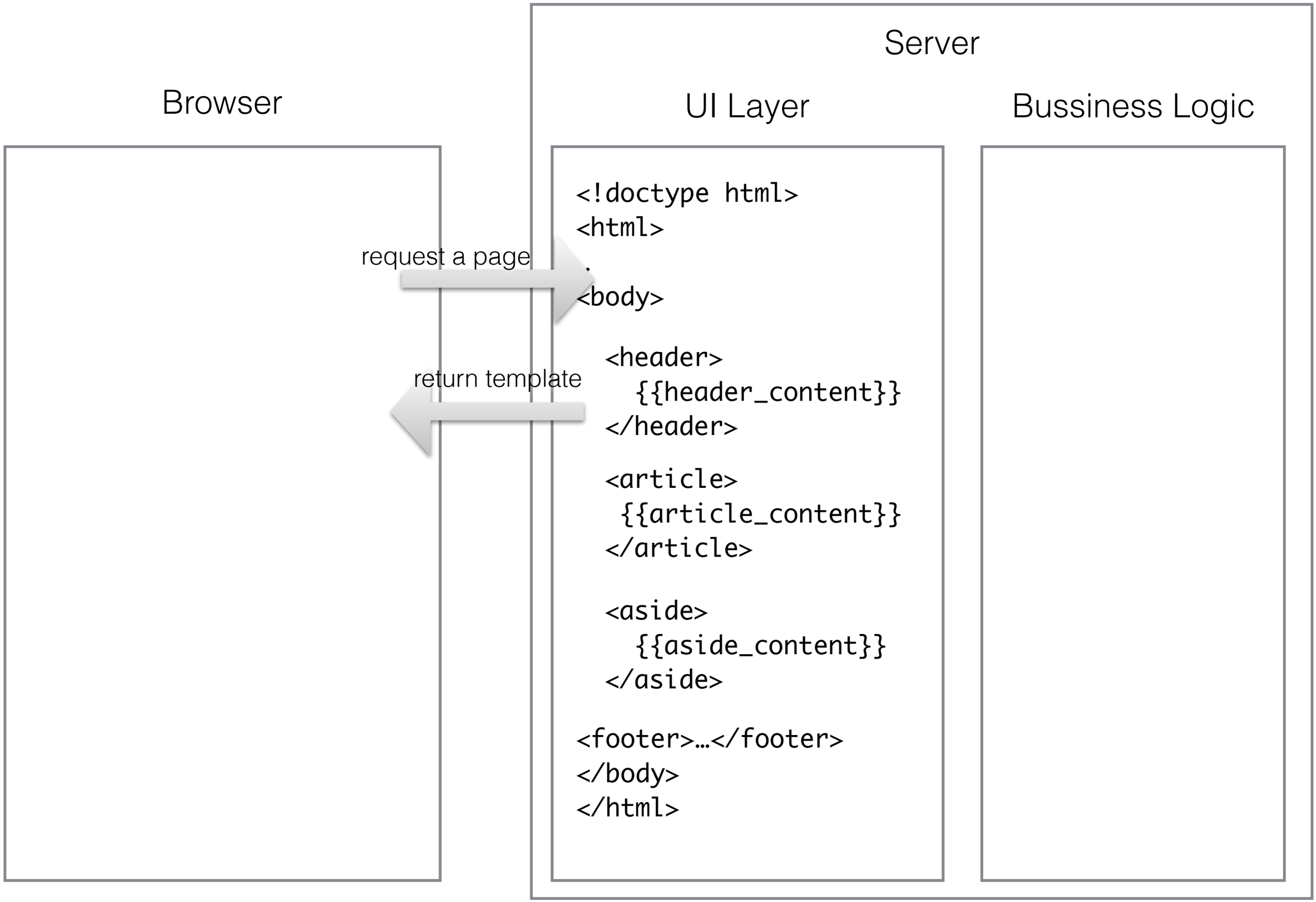
So Here's the BigPipe Way of Page Initialization



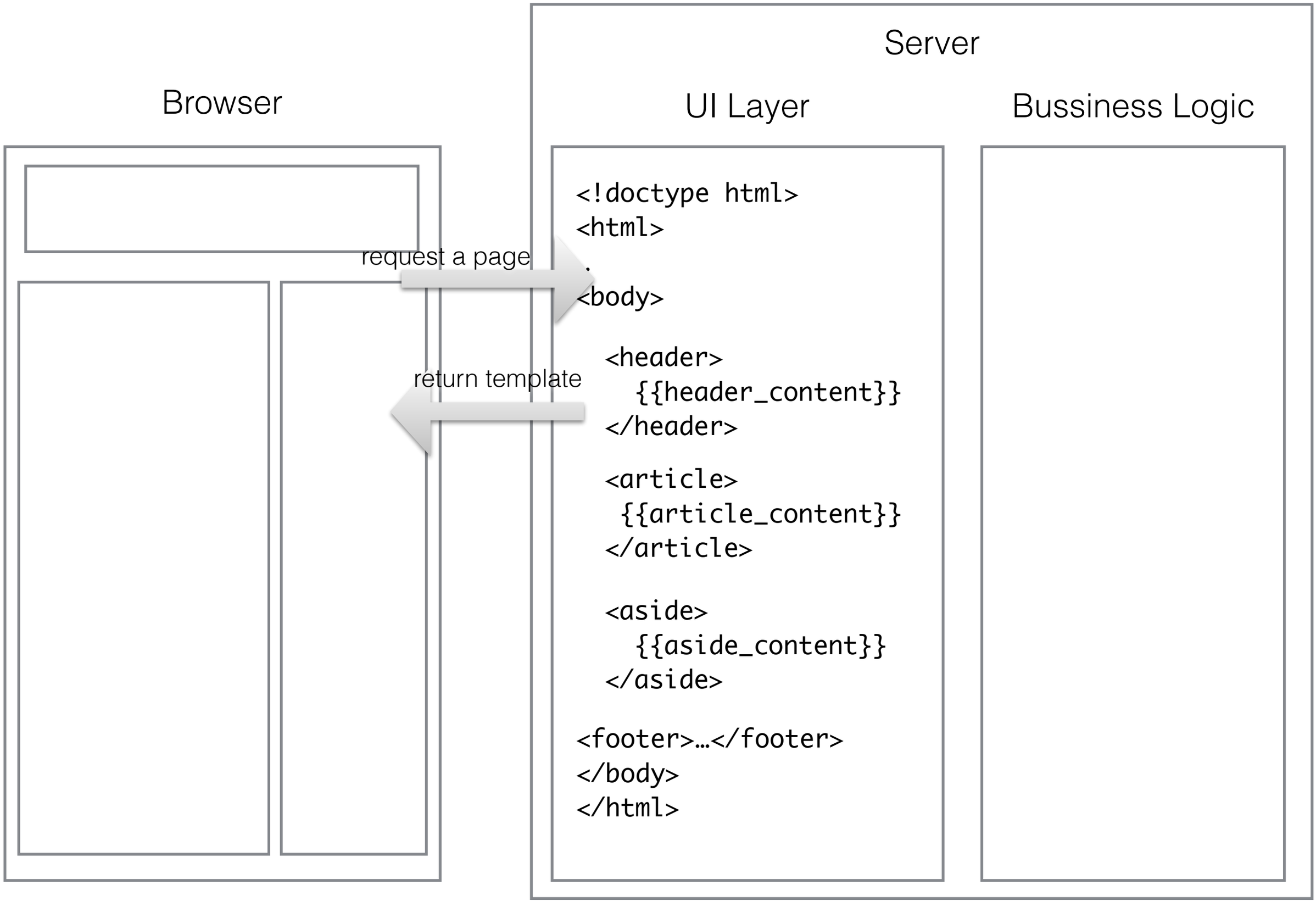
So Here's the BigPipe Way of Page Initialization



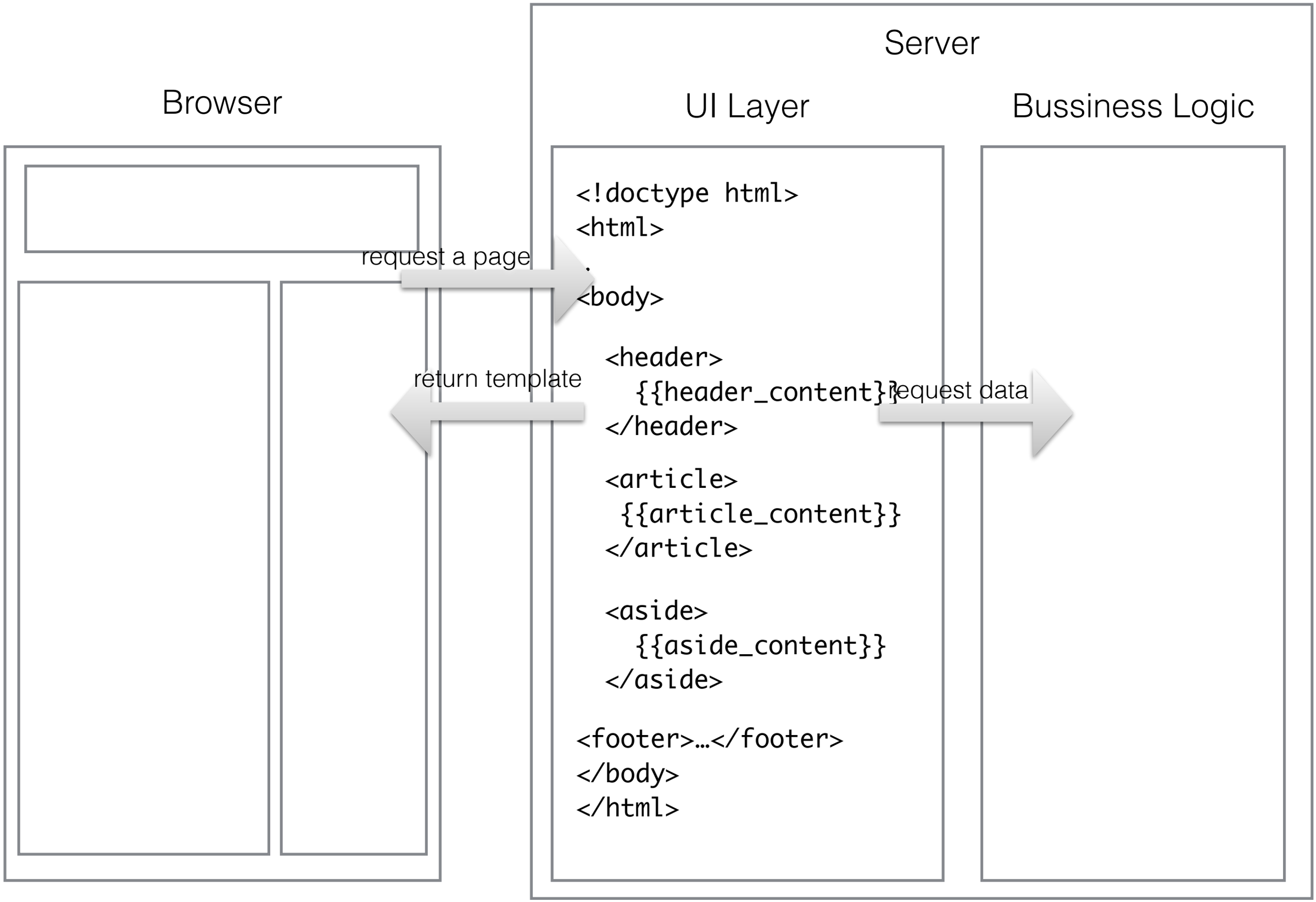
So Here's the BigPipe Way of Page Initialization



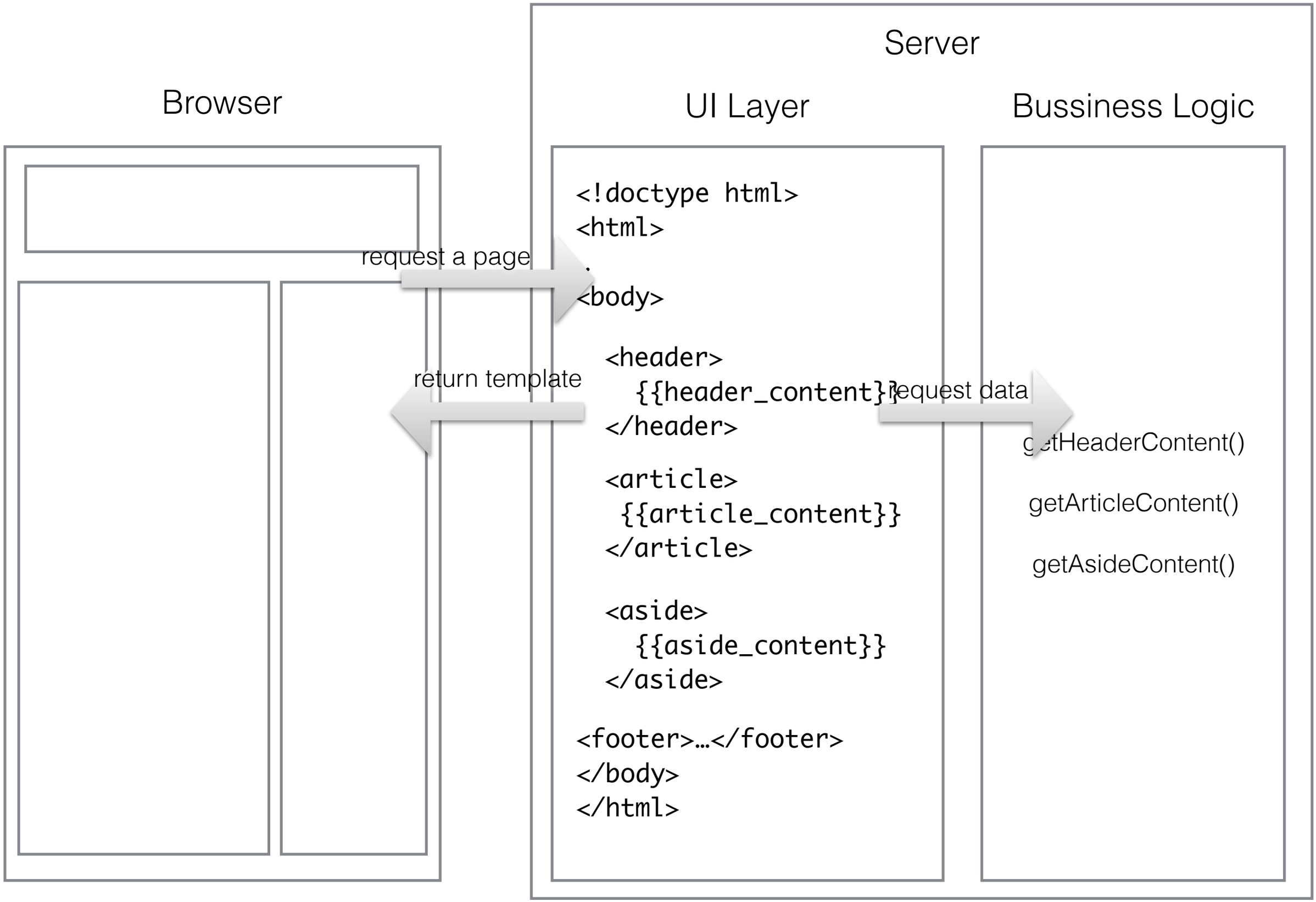
So Here's the BigPipe Way of Page Initialization



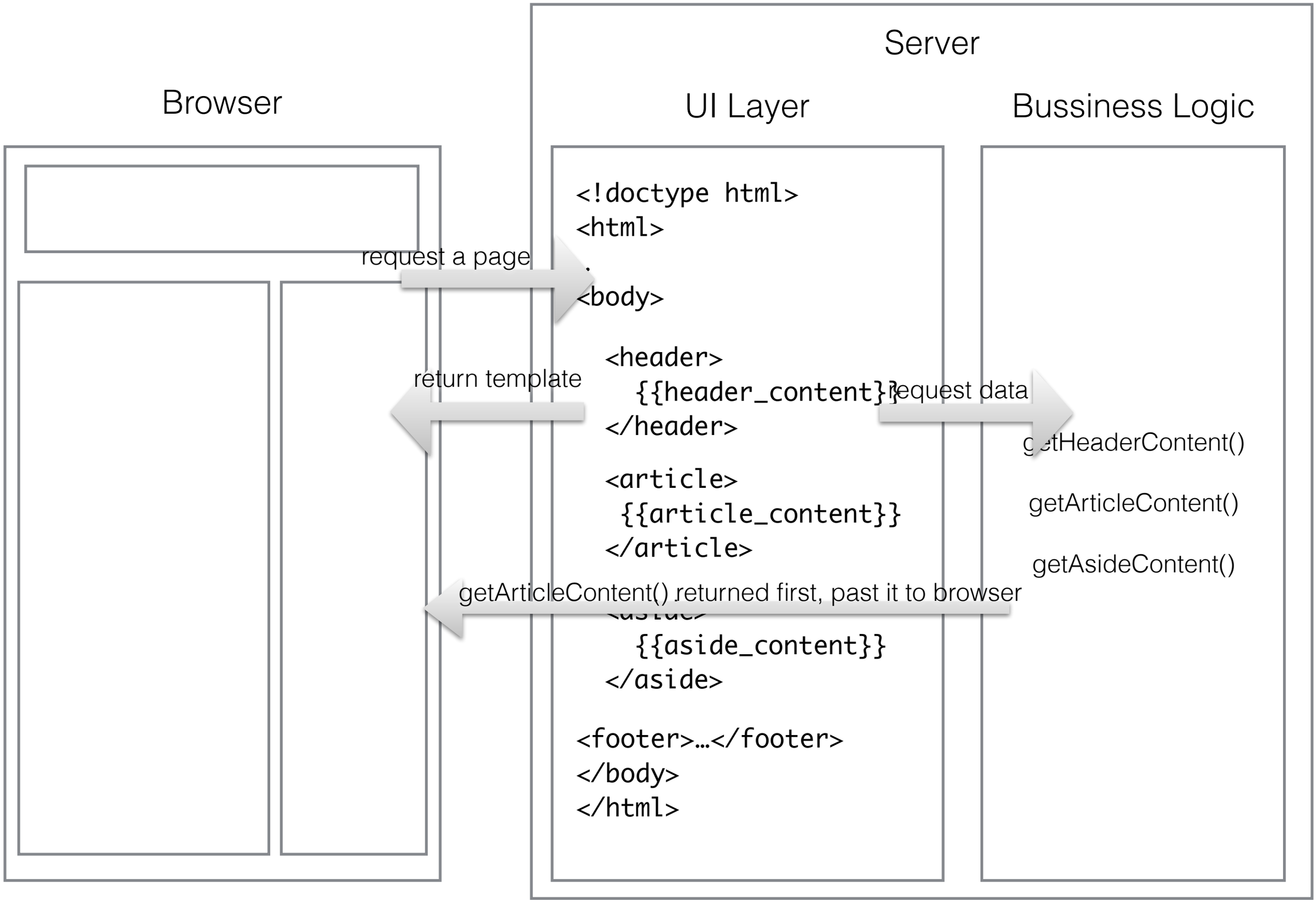
So Here's the BigPipe Way of Page Initialization



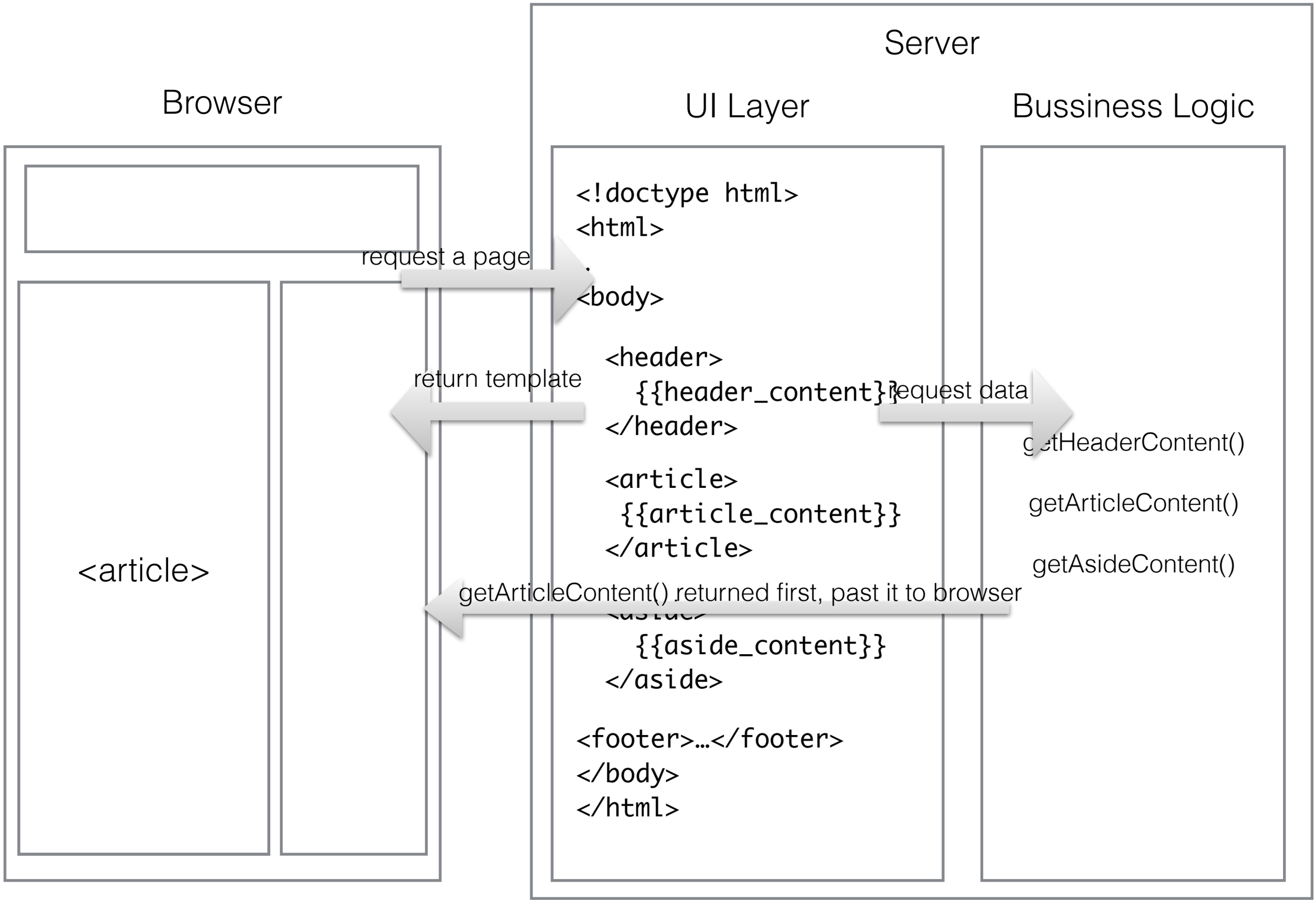
So Here's the BigPipe Way of Page Initialization



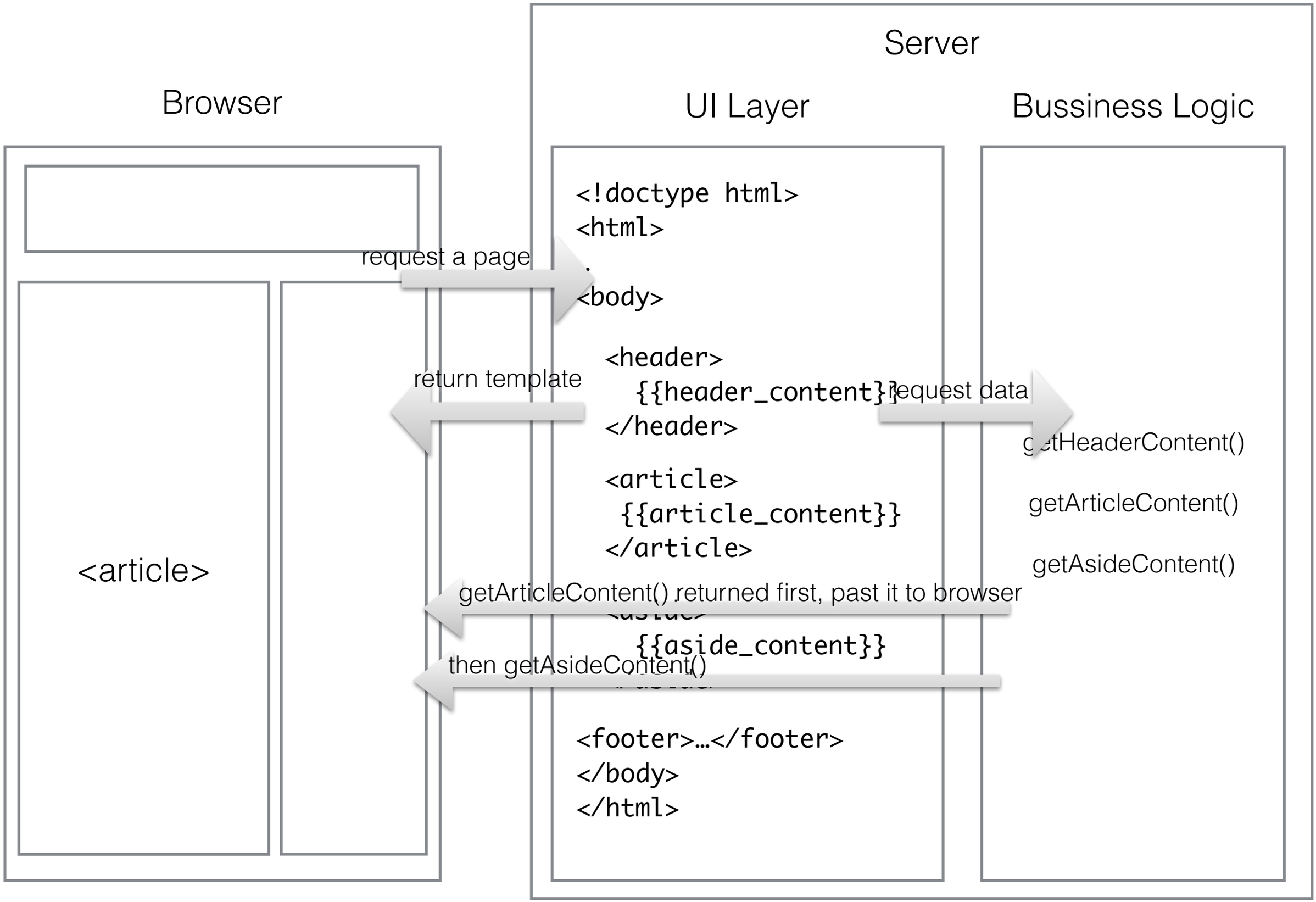
So Here's the BigPipe Way of Page Initialization



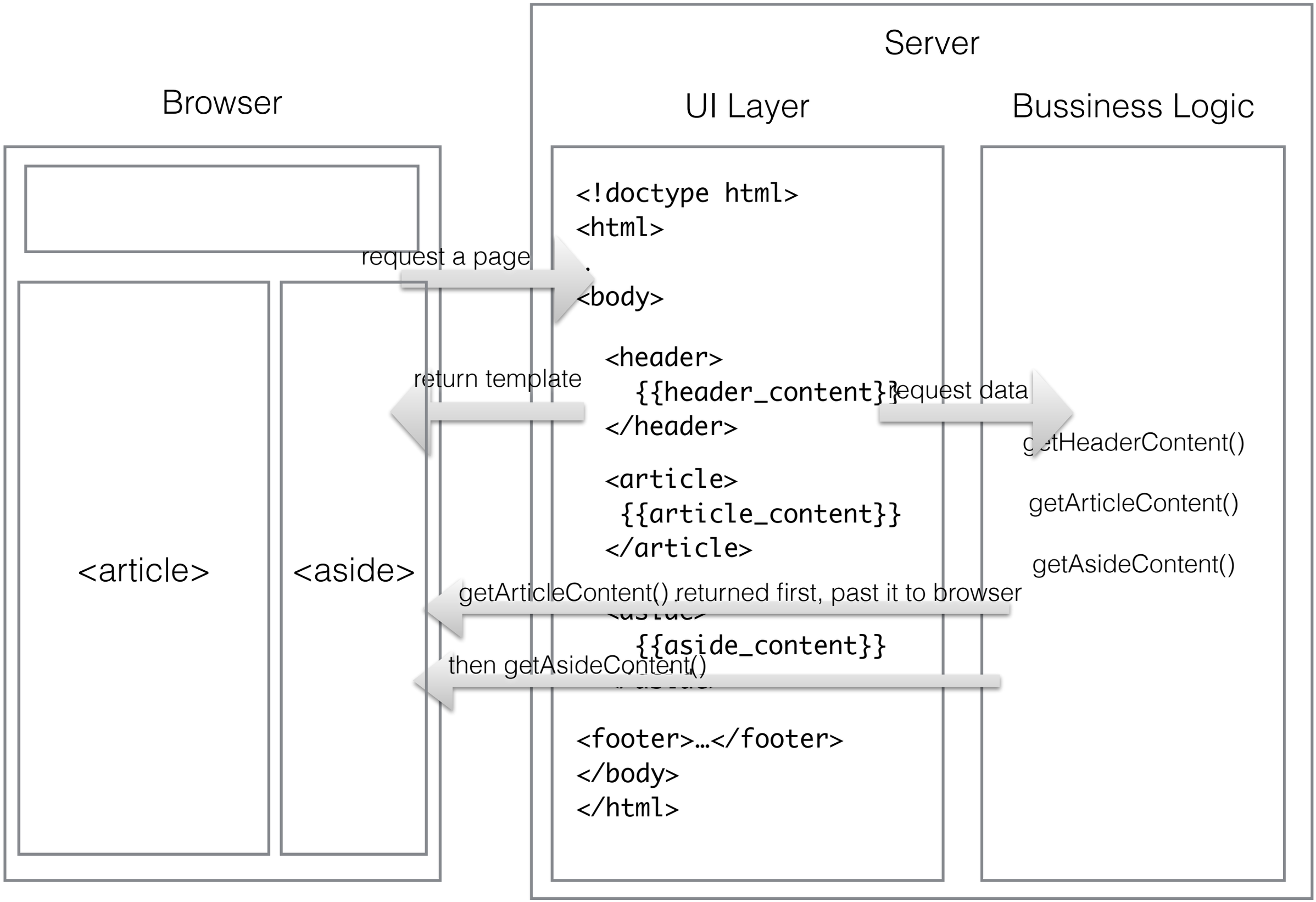
So Here's the BigPipe Way of Page Initialization



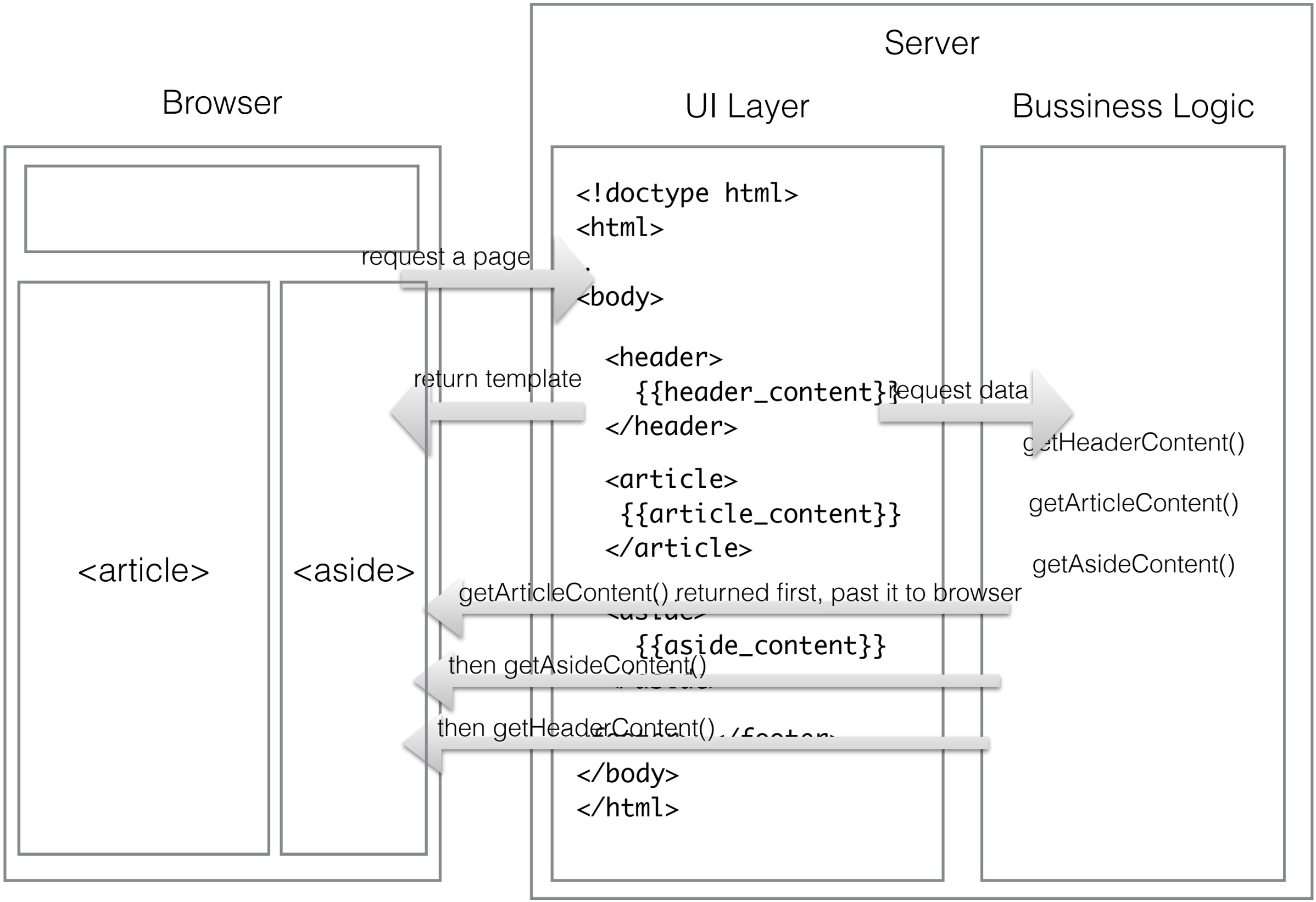
So Here's the BigPipe Way of Page Initialization



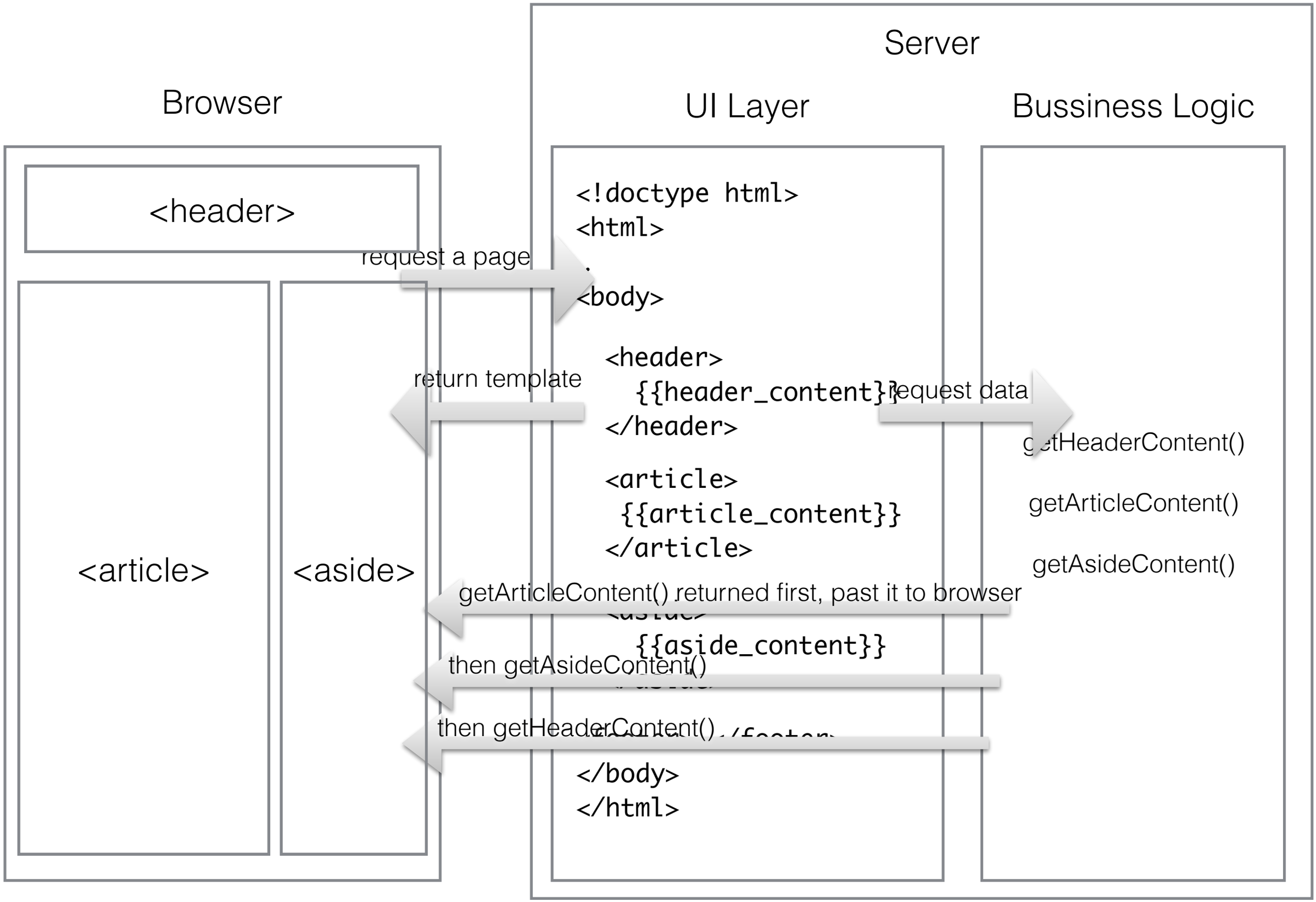
So Here's the BigPipe Way of Page Initialization



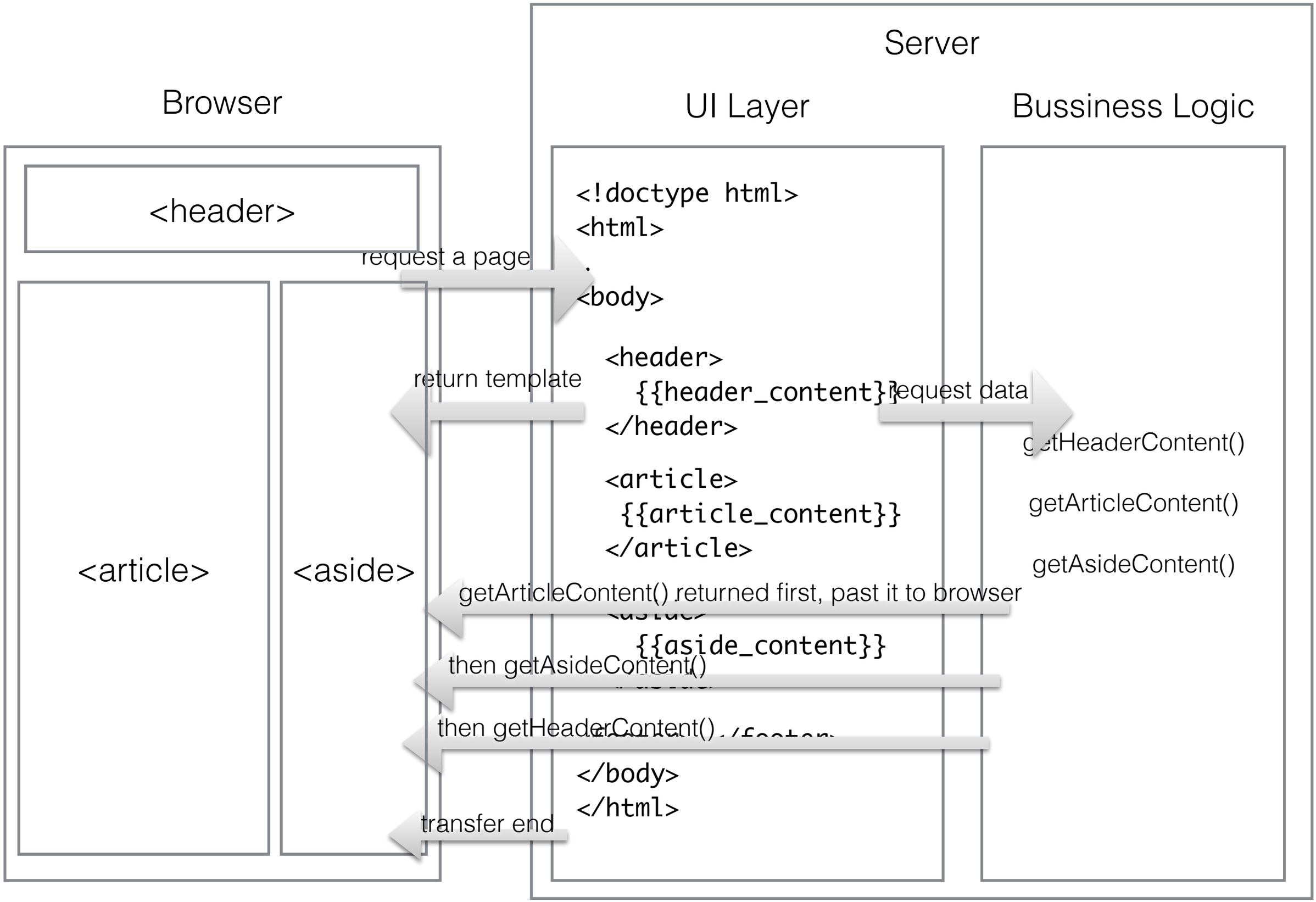
So Here's the BigPipe Way of Page Initialization



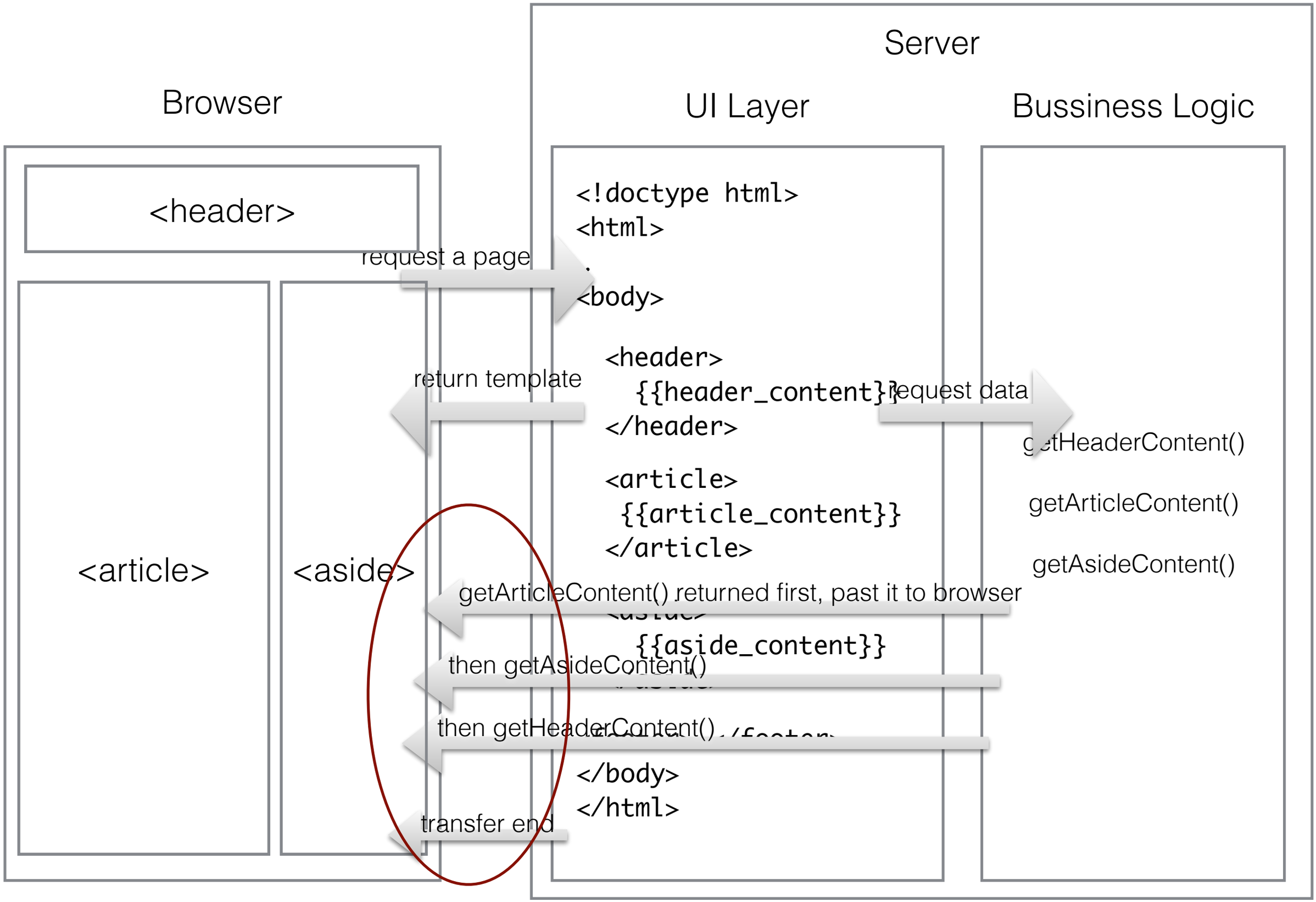
So Here's the BigPipe Way of Page Initialization



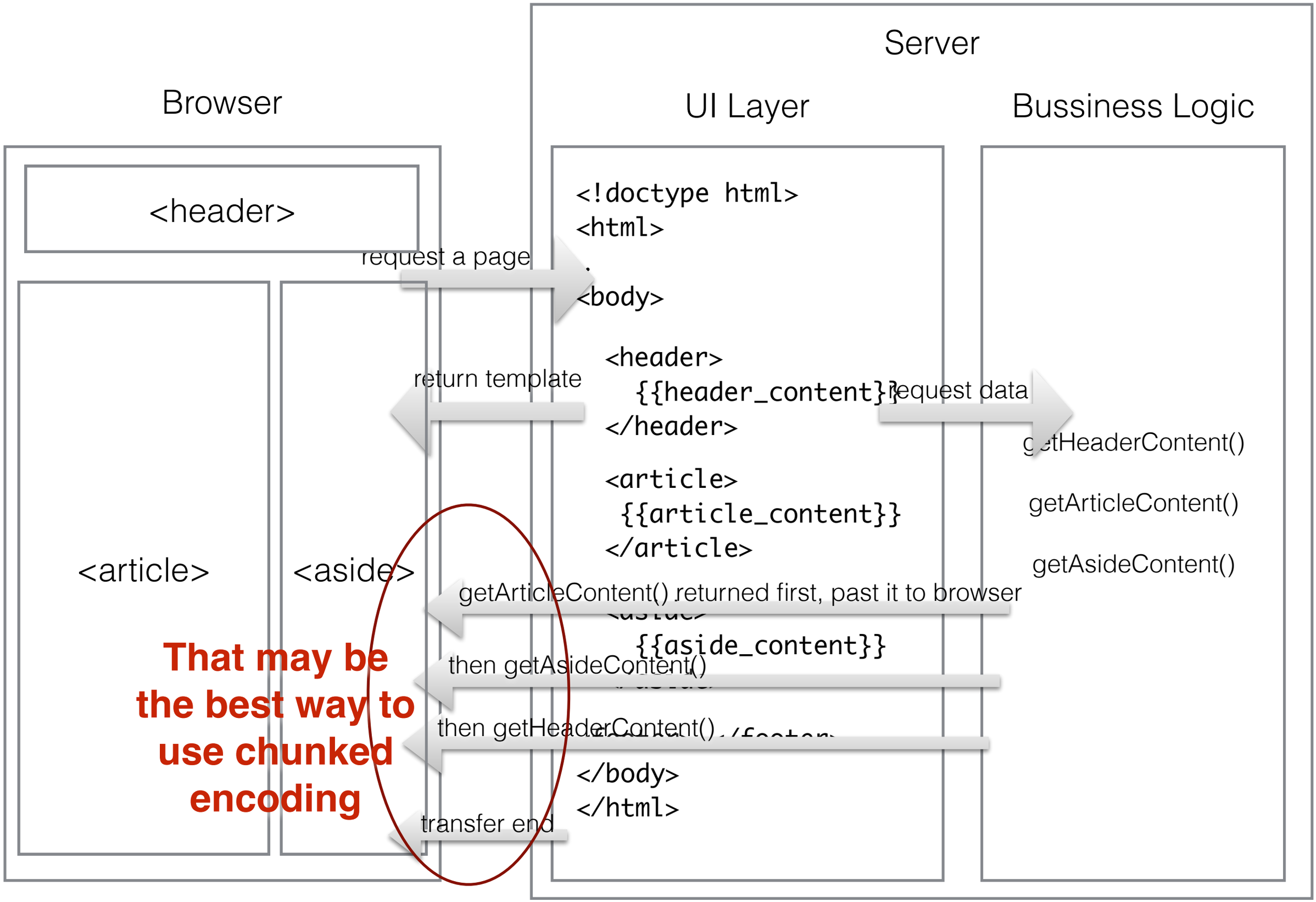
So Here's the BigPipe Way of Page Initialization



So Here's the BigPipe Way of Page Initialization



So Here's the BigPipe Way of Page Initialization



On “BigPipe on Node”

A “Simple” Demo - Sorry I couldn’t make it simpler

bigpipe.js (~/b

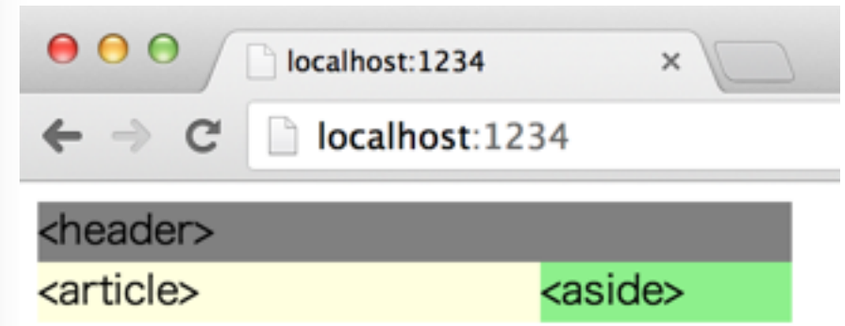
```
var http = require('http');
http.createServer(function (req, res) {
  res.setHeader('Content-Type', 'text/html');
  res.write('<style>header{width:300px;background:gray;}</style>');
  res.write('<style>article{width:200px;background:lightyellow;float:left;}</style>');
  res.write('<style>aside{width:100px;background:lightgreen;float:left;}</style>');
  res.write('<header></header><article></article><aside></aside>');
  res.write('<script>function insertHtml(tag, html){ document.getElementsByTagName('+
    'tag)[0].innerHTML = html;}</script>');
  var remain = 3;
  setTimeout(function () {
    res.write('<script>insertHtml("header", "&lt;header&gt;")</script>');
    sendEnd(--remain);
  }, 5000);
  setTimeout(function () {
    res.write('<script>insertHtml("article", "&lt;article&gt;")</script>');
    sendEnd(--remain);
  }, 1000);
  setTimeout(function () {
    res.write('<script>insertHtml("aside", "&lt;aside&gt;")</script>');
    sendEnd(--remain);
  }, 3000);
  function sendEnd(r) {
    if (r) return;
    else res.end();
  }
}).listen(1234);
```

More example: http://github.com/undoZen/bigpipe-on-node

Result

2. bash

```
Connected to localhost.
Escape character is '^]'.
GET / HTTP/1.1
Host: localhost

HTTP/1.1 200 OK
Content-Type: text/html
Date: Fri, 08 Nov 2013 16:46:36 GMT
Connection: keep-alive
Transfer-Encoding: chunked

33
<style>header{width:300px;background:gray;}</style>
46
<style>article{width:200px;background:lightyellow;float:left;}</style>
43
<style>aside{width:100px;background:lightgreen;float:left;}</style>
33
<header></header><article></article><aside></aside>
69
<script>function insertHtml(tag, html){ document.getElementsByTagName(tag)[0].in
nerHTML = html;}</script>
39
<script>insertHtml("article", "&lt;article&gt;")</script>
35
<script>insertHtml("aside", "&lt;aside&gt;")</script>
37
<script>insertHtml("header", "&lt;header&gt;")</script>
0

Connection closed by foreign host.
~ - @undozen
$
```



Why Node for BigPipe

Why Node for BigPipe

- It’s JavaScript on the server-side

Why Node for BigPipe

- It's JavaScript on the server-side
  - So templates could be (re)used on both server-side and browser-side

Why Node for BigPipe

- It's JavaScript on the server-side
  - So templates could be (re)used on both server-side and browser-side
  - So you can fallback to Content-Length way

Why Node for BigPipe

- It's JavaScript on the server-side
  - So templates could be (re)used on both server-side and browser-side
    - So you can fallback to Content-Length way
- It response HTTP request using chunked encoding <u>by default</u>

On Node

Why Node for Us
(as front-end developers)

Why Node for Us (as front-end developers)

- HTML generation should be front-end developers' consideration
  (while back-end developers are focusing on data)

Why Node for Us (as front-end developers)

- HTML generation should be front-end developers' consideration
  (while back-end developers are focusing on data)

- for me, it's the most natural way to co-operate with back-end developer

In Traditional Company…

Front-end Team

Browser

<header>

<article>

<aside>

Back-end Team

Server (Python/Ruby/Java)

UI Layer

```
<!doctype html>
<html>
…
<body>

  <header>
    {{header_content}}
  </header>

  <article>
   {{article_content}}
  </article>

  <aside>
    {{aside_content}}
  </aside>

<footer>…</footer>
</body>
</html>
```

Bussiness Logic

getHeaderContent()

getArticleContent()

getAsideContent()

In Traditional Company…

Front-end Team

Browser

<header>

<article>

<aside>

Back-end Team

Server (Python/Ruby/Java)

UI Layer

```
<!doctype html>
<html>
…
<body>

  <header>
    {{header_content}}
  </header>

  <article>
   {{article_content}}
  </article>

  <aside>
    {{aside_content}}
  </aside>

<footer>…</footer>
</body>
</html>
```

Bussiness Logic

getHeaderContent()

getArticleContent()

getAsideContent()

In Traditional Company…

Front-end Team

Back-end Team

Server (Python/Ruby/Java)

Browser

<header>

<article>

<aside>

UI Layer

```
<!doctype html>
<html>
…
<body>

  <header>
    {{header_content}}
  </header>

  <article>
   {{article_content}}
  </article>

  <aside>
    {{aside_content}}
  </aside>

<footer>…</footer>
</body>
</html>
```

Bussiness Logic

getHeaderContent()

getArticleContent()

getAsideContent()

In Traditional Company…

Front-end Team

Back-end Team

Browser

<header>

<article>

<aside>

Server (Python/Ruby/Java)

UI Layer

```
<!doctype html>
<html>
…
<body>

  <header>
    {{header_content}}
  </header>

  <article>
   {{article_content}}
  </article>

  <aside>
    {{aside_content}}
  </aside>

<footer>…</footer>
</body>
</html>
```

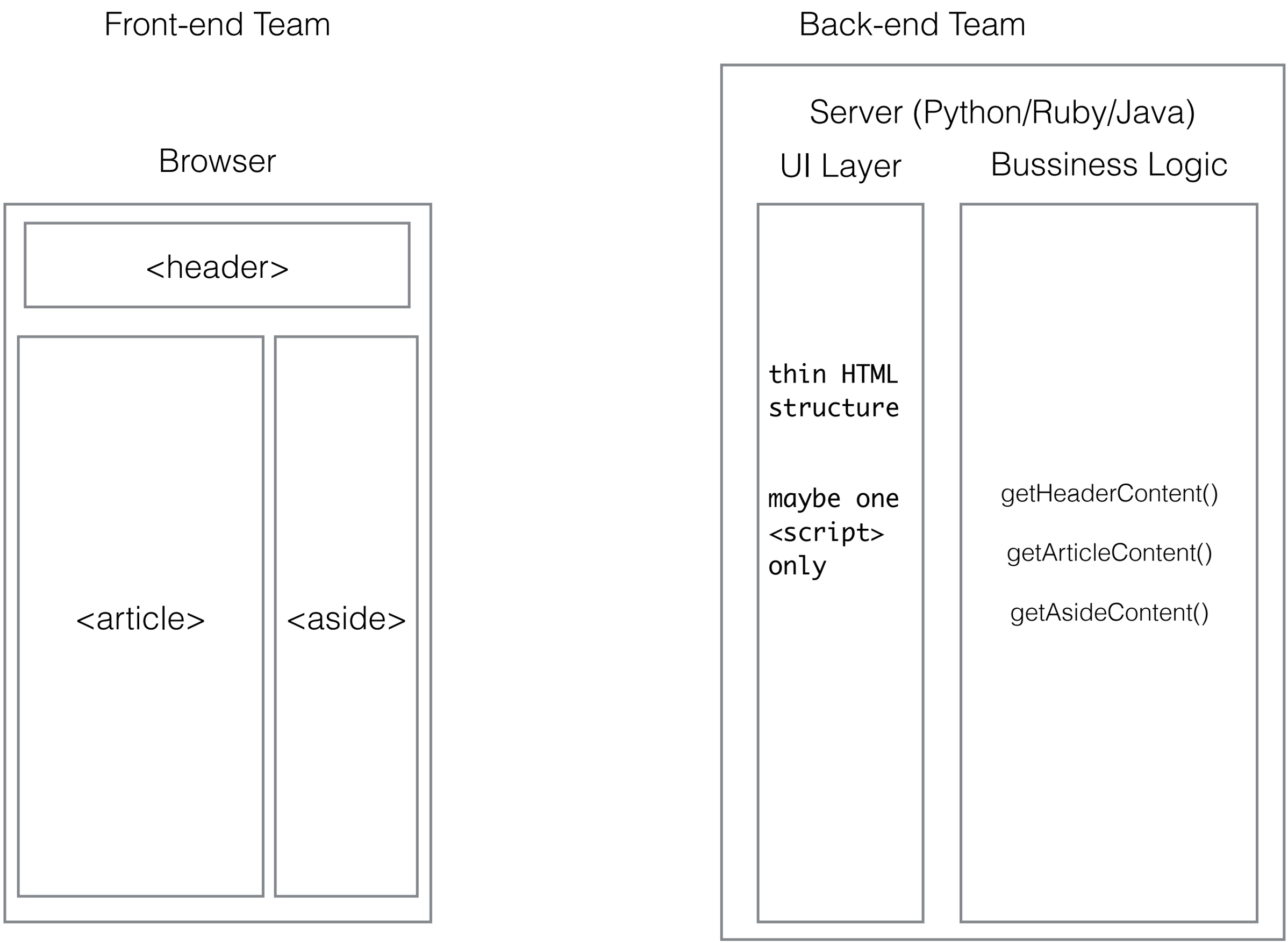
Bussiness Logic

getHeaderContent()

getArticleContent()

getAsideContent()

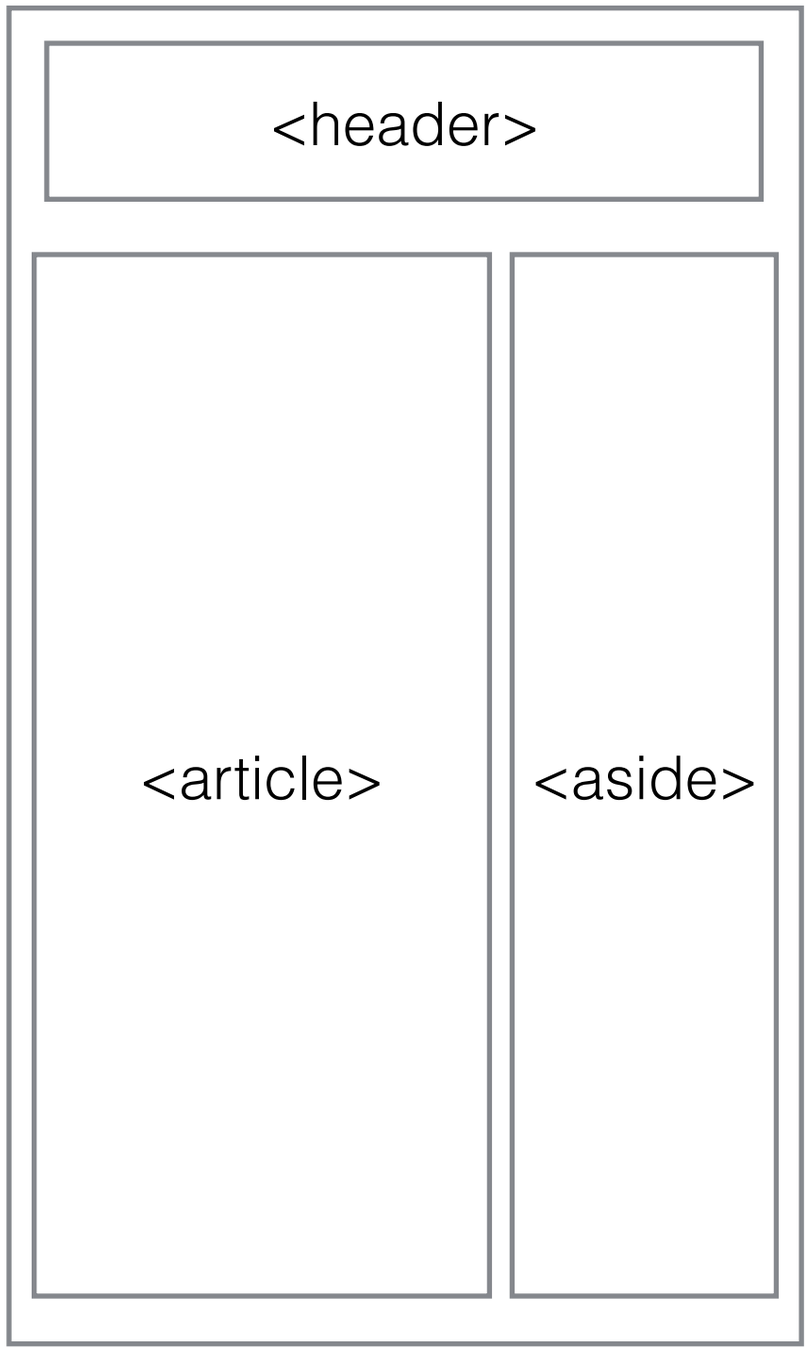
In Heavy Web App Team…



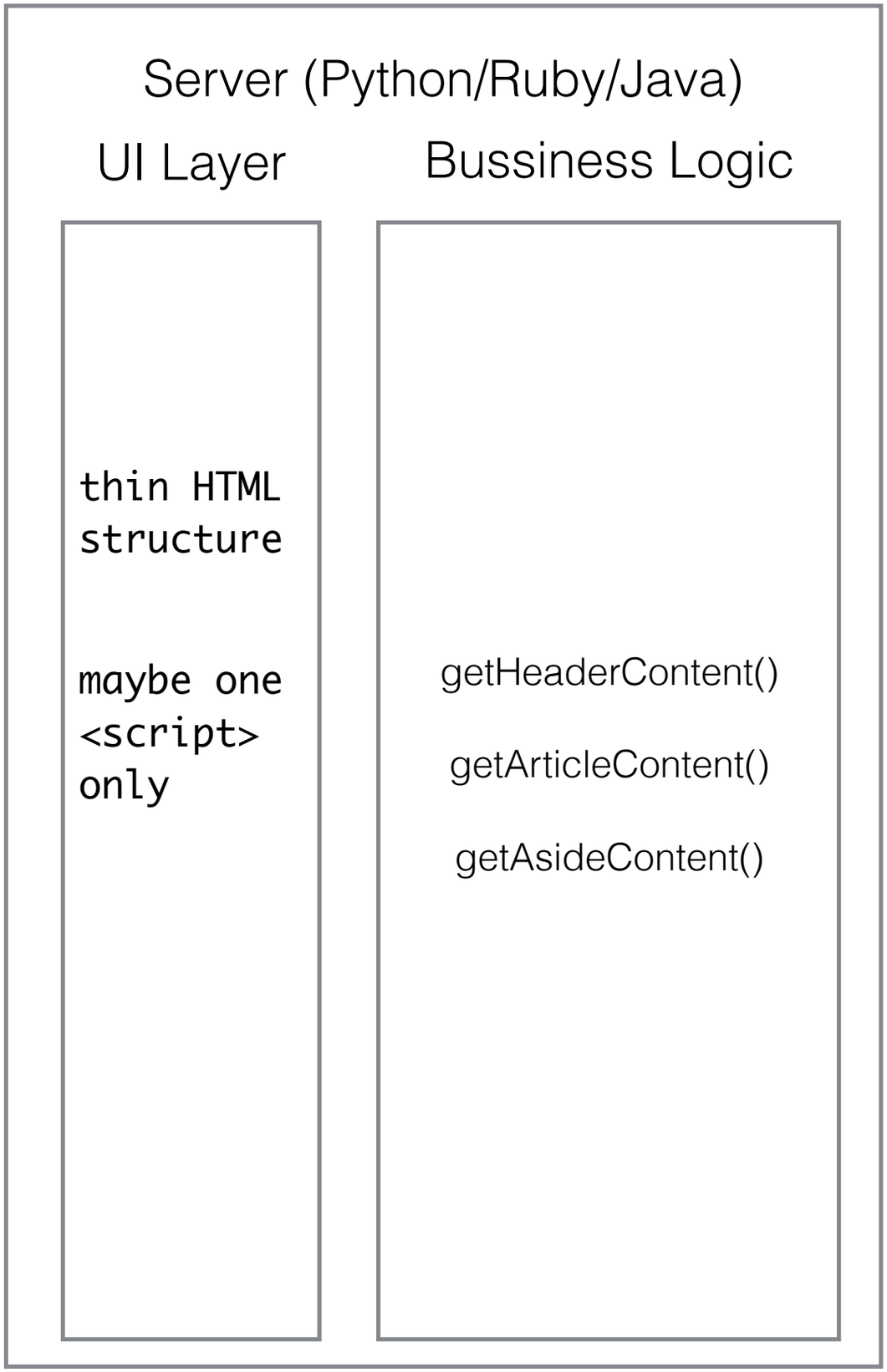
In Heavy Web App Team…

Front-end Team

Browser



Back-end Team



In Company Using Node.js…

Front-end Team

Back-end Team

Node.js

Browser

UI Server

Data Server

<header>

<article>

<aside>

```
<!doctype html>
<html>
…
<body>

  <header>
    {{header_content}}
  </header>

  <article>
   {{article_content}}
  </article>

  <aside>
    {{aside_content}}
  </aside>

<footer>…</footer>
</body>
</html>
```

getHeaderContent()

getArticleContent()

getAsideContent()

In Company Using Node.js…

Front-end Team

Back-end Team

Node.js

Browser

UI Server

Data Server

<header>

<article>

<aside>

```
<!doctype html>
<html>
…
<body>

  <header>
    {{header_content}}
  </header>

  <article>
   {{article_content}}
  </article>

  <aside>
    {{aside_content}}
  </aside>

<footer>…</footer>
</body>
</html>
```

getHeaderContent()

getArticleContent()

getAsideContent()

In Some Awesome（牛逼） or Awful（苦逼） Company…

Full-Stack Team

Whatever ┐(´ ∇ `)┌

Browser

<header>

<article>

<aside>

UI Layer

```
<!doctype html>
<html>
…
<body>

  <header>
    {{header_content}}
  </header>

  <article>
   {{article_content}}
  </article>

  <aside>
    {{aside_content}}
  </aside>

<footer>…</footer>
</body>
</html>
```

Bussiness Logic

getHeaderContent()

getArticleContent()

getAsideContent()

What I perfer more…

Front-end Team

Back-end Team

Node.js

Browser

UI Server

Data Server

<header>

<article>

<aside>

```
<!doctype html>
<html>
…
<body>

  <header>
    {{header_content}}
  </header>

  <article>
   {{article_content}}
  </article>

  <aside>
    {{aside_content}}
  </aside>

<footer>…</footer>
</body>
</html>
```

getHeaderContent()

getArticleContent()

getAsideContent()

Referring

- The Original Facebook Article About BigPipe
  www.facebook.com/note.php?note_id=389414033919

- My Chinese Article
  https://github.com/undoZen/bigpipe-on-node

- Another Chinese Article
  www.searchtb.com/2011/04/an-introduction-to-bigpipe.html

- N. C. Zakas on Node.js
  www.nczonline.net/blog/2013/10/07/node-js-and-the-new-web-front-end/

Thank You!

You Can Find Me on the Internet

- 王宇 Anson Yu Wang
- Twitter.com/undoZen GitHub.com/undoZen
- Weibo.com/n/JavaScript发烧友
- http://www.undozen.com
- mailto: hello@undozen.com